Context-Aware and Location Systems

Giles John Nelson

Clare College



A dissertation submitted for the degree of Doctor of Philosophy in the University of Cambridge

January 1998

To my mother and father

Abstract

Computing systems are becoming increasingly mobile, with users interacting with many networked devices, both fixed and portable, over the course of a day. This has lead to the emergence of a new paradigm of computing, the *context-aware* system. A user's context can be described as their relationship to computing devices, colleagues and the surrounding physical environment. Applications using such a system are made aware of changes in the physical and computing environment, and adjust themselves accordingly. For example, with knowledge of the locations of people and equipment, a suitable nearby display can be determined to deliver a message to a mobile user.

Context-aware systems are currently ad-hoc in nature, typically developed around a particular technology and oriented towards supporting a particular type of application. Current location systems are similarly ad-hoc with no notion of abstraction above a particular locating technology. Furthermore, little analysis has been conducted into the application requirements of a location system. This general lack of systems design makes current approaches difficult to extend and generalise.

This thesis proposes a systems architecture for the support of context-aware applications. It is asserted that such an architecture requires three main areas of capability. Firstly, a context-aware system needs to monitor the physical environment and make this information available in a uniform way. Secondly, location information requires higher-level management together with a representation of static parts of the physical domain. Further, support is required to enable complex, asynchronous monitoring of spatial areas. Thirdly, application support is required to access distributed sensor and location services, so facilitating the simple specification of context-aware scenarios.

Preface

I would like to thank my supervisor, Andy Hopper, for his encouragement, patience and support, and for enabling me to study in Cambridge.

Numerous members of the Computer Laboratory have given me help and advice. John Bates has given much encouragement and practical help, often beyond the call of duty. I am grateful to Andy Ward and John Naylon for putting their own work at my disposal. Anthony Rowstron has given valuable comments on drafts of this dissertation. Jean Bacon and Ken Moody have provided experience and insight at various times. Lewis Tiffany provided expert librarianship.

I am thankful to ORL for providing me with a CASE studentship as well as an extra year of funding. Andy Harter has provided much encouragement and support. Frazer Bennett, Alan Jones, Frank Stejano, and many others have given me help at various times.

This work would not have been possible without the support of my family, particularly my mother and my late father. I must also thank my fiancée, Helen, for her love and empathic help. Words of encouragement are immensely valuable when undertaking a difficult task. Numerous friends, too many to mention here, have provided these over the years. I am thankful to them all.

I am grateful to the Engineering and Physical Sciences Research Council for providing three years of funding.

This dissertation is the result of my own work and is not the outcome of any work done in collaboration. I declare that this dissertation is not the same as any other dissertation I have submitted for a degree, diploma or other qualification at any university. Furthermore, no part of this dissertation has been or is currently being submitted for any such qualification.

This dissertation does not exceed sixty thousand words, including tables, footnotes and bibliography.

Contents

Li	List of Figures xviii								
Li	List of Tables xix								
1	Intr	oduction	1						
	1.1	Context-Awareness	1						
	1.2	Location Information	2						
	1.3	Environmental Awareness	3						
	1.4	Target Environment	4						
	1.5	Research Aims	4						
	1.6	Dissertation Outline	5						
2	Rel	ted Work	7						
	2.1	Location Technologies	7						
		2.1.1 Active Badges	7						
		2.1.2 Electromagnetic Tracking	8						
		2.1.3 Optoelectronic Tracking	8						
		2.1.4 GPS	9						
	2.2	Location Information Management	9						
		2.2.1 Dataman	9						
		2.2.2 Hierarchical Location Management	10						
		2.2.3 A Location Service Design	11						
		2.2.4 Integration of Location Hints	11						
		2.2.5 Active Maps	12						
		2.2.6 GIS	13						
	2.3	Environmental Sensing	13						
		2.3.1 The Interactive Office	13						

		2.3.2	Monitoring Computer Use	14
		2.3.3	An Application Interface	14
		2.3.4	SPIRIT	14
	2.4	Ubiqui	tous Computing	15
	2.5	Augme	ented Reality	16
		2.5.1	Knowledge Based Generation	16
		2.5.2	Wearable Computing	16
		2.5.3	Situated Information Spaces	16
	2.6	Contex	xt-Aware Applications	17
		2.6.1	Teleporting	17
		2.6.2	Mobile Agents	18
		2.6.3	Stick-e Documents	18
		2.6.4	Memory Protheses	18
		2.6.5	Computing Personae	19
	2.7	Event	Management and Composition	20
		2.7.1	Distributed Events	20
, ,	2.8	Review	Ϋ	20
0				0.0
.	Ana	uysis a	nd Requirements	23
	3.1	Reviev	v and Critique of Related Work	23
		3.1.1	Applications	23
		3.1.2 9.1.9	Sensor Systems	24
		3.1.3 9.1.4	Climit Line C	24
	9.9	3.1.4 CALA	Client Interfaces	25
•	3.2	CALA	15 Requirements	25
		3.2.1 2.2.2	Sensor System Interfaces	25 96
		3.2.2	Location Information Management	26 26
		3.2.3	Provision of Persistent Resources	26 26
		3.2.4 2.2.5		20
	<u>.</u>	3.2.5 Cl	Performance	27
	3.3	Chapt	er Summary	27
!	\mathbf{Sen}	sor Sys	stems	29
4	4.1	Active	Badge System	29

	4.2	Ultras	ound Badge System
		4.2.1	Deployment
	4.3	Works	tation Activity Monitoring
		4.3.1	Monitoring X-Windows Activity
		4.3.2	How xmonitor Works
		4.3.3	Deployment
	4.4	Door	Position
		4.4.1	Providing Door Status Information
		4.4.2	Deployment
	4.5	Telepł	none Handsets
		4.5.1	Deployment
	4.6	Motio	n Detection
		4.6.1	Detecting Motion from Video
		4.6.2	Deployment
	4.7	Summ	ary
5	\mathbf{Eve}	nt Ser	vice Architecture 39
	5.1	Events	s and Event Service Requirements
		5.1.1	Event Definition
		5.1.2	Event Service Requirements
	5.2	A Sta	ndard Distributed Architecture
		5.2.1	Inter-ORB Communication
		5.2.2	Local System Setup 42
	5.3	The S	tructure of CALAIS Event Services
		5.3.1	Registration and Call-back
		5.3.2	Time-stamps
		5.3.3	Storing and Querying Events
		5.3.4	Dynamically Discovering an Event Interface 45
	5.4	Buildi	ng Event Services
	5.5	D 1	ase Support Services
	r c	Datab	
	0.0	Datab Comp	onent Summary 48
	5.0 5.7	Datab Comp Distri	onent Summary48buted Time48
	5.0 5.7	Datab Comp Distri 5.7.1	onent Summary 48 buted Time 48 Clock Synchronisation 49
	5.7	Datab Comp Distri 5.7.1 5.7.2	onent Summary 48 buted Time 48 Clock Synchronisation 49 Network and Registration Delay 50

		5.8.1	Inter-Domain Working
	5.9	Summ	nary
6	Loc	ation S	Service Requirements 53
	6.1	The S	ignificance of Location
	6.2	Requi	rements
		6.2.1	Location Information Integration
		6.2.2	Physical Object Representation
		6.2.3	Types of Query
		6.2.4	Orientation $\ldots \ldots 56$
		6.2.5	Performance and Scale
		6.2.6	CALAIS Integration
	6.3	A Spa	tially Equipped DBMS
7	Too	ation (Service Design
1	LOC	Physic	cal Object Representation 62
	(.1	7 1 1	Handling Location Errors 63
		719	Record Attributes 63
	79	Indevi	ing Technologies 64
	1.2	7 2 1	Rester 64
		7.2.1	Point-Based 65
		723	Complex Objects - Rectangles 65
	7.3	Locati	ion Service Core
	1.0	731	Indexing 67
		732	Coordinate Systems 67
		733	Interfacing to Location Sources 67
		7.3.4	Performing an Update
	7.4	Repre	senting Orientation
		7.4.1	Expressing Orientation
	7.5	Query	
		7.5.1	Spatial Queries
		7.5.2	Name Queries 72
		7.5.3	Summary
	7.6	Repre	senting the Environment
		7.6.1	Representation of Rooms

		7.6.2	Internal Representation
		7.6.3	Searching by Container Name
		7.6.4	Communication Between Containers
		7.6.5	Environment Database
	7.7	Appli	cation Interface
		7.7.1	Update Interface
		7.7.2	Search by Name Interface
		7.7.3	Registration Interface
	7.8	Summ	nary
8	Loc	ation	Service Evaluation 83
	8.1	The L	ocation Service in Use
		8.1.1	Representing the Location Domain
		8.1.2	Location Sources
		8.1.3	Database Resources
		8.1.4	Query Evaluation
		8.1.5	An Extended Example - Promiscuous Teleporting 92
		8.1.6	Summary
	8.2	Locat	ion Service Indexing Performance
		8.2.1	Determining the Size of an Indexing Node 96
		8.2.2	Object Density
		8.2.3	Index Size
		8.2.4	Searching Performance
		8.2.5	Summary
	8.3	Locat	ion Service Performance
		8.3.1	Updating
		8.3.2	Evaluating Queries
		8.3.3	Performance Conclusions
9	Cor	nposit	e Event Management 103
	9.1	Motiv	ations $\ldots \ldots 103$
	9.2	Requi	rements
	9.3	A Lar	nguage for Expressing Composite Events
		9.3.1	Basic Operators
		9.3.2	Operator Precedence

		9.3.3	Variable Instantiation and Matching	106	
	9.4	Param	eter Contexts	108	
	9.5	State I	Jookups	111	
	9.6	Timers	3	112	
		9.6.1	Use of Timers with State Lookups $\ . \ . \ . \ . \ .$.	113	
	9.7	Operat	ting Within a Distributed Environment	113	
		9.7.1	Distributed Time	113	
		9.7.2	Network Delays	113	
		9.7.3	Registration Delays	114	
	9.8	Implen	nentation Considerations	114	
		9.8.1	Application Interface	114	
		9.8.2	Initialisation	115	
		9.8.3	CE Monitoring	115	
	9.9	Compo	osite Event System Evaluation	116	
		9.9.1	$Composite \ Event \ Examples \ \ \ldots $	116	
		9.9.2	An Extended Example	119	
		9.9.3	CE System Performance	121	
		9.9.4	Wider Application of Composite Events	121	
	9.10	Summa	ary	121	
10	Con	clusior	15	123	
	10.1	Summa	ary	123	
	10.2	Potential Further Work			
	-				
A Event Source IDL		nt Sou		133	
	A.1	Generi	c Interfaces (register.idl)	133	
	A.2 Active Badge		Badge	133	
	A.3	Ultrase		134	
	A.4	WORKS		134	
	A.3	Door M		130	
	A.0	Teleph	one Handset Monitoring	130	
	А.(INFOLIOI		190	
В	\mathbf{Con}	tainer	and Junction Database Entries	137	
	B.1	ORL F	Floor 1 Containers	137	
	B.2	ORL F	Floor 1 Junctions	137	

List of Figures

2.1	Server Hierarchy 10
3.1	CALAIS Architecture Overview
4.1	Layout of Badge Sensor Network
4.2	Ultrasound Badge System
4.3	xmonitor and Marshaller Structure
4.4	Door Status Service Structure
4.5	Typical Medusa Setup
4.6	Motion Detection System
5.1	Generic Structure of an Event Service
5.2	Structure of an Event Service
5.3	Dynamically Discovering an Event Class
5.4	Building a CALAIS Event Service from Components 47
5.5	CALAIS Components 49
6.1	Position and Error Representation in a Container
7.1	Proposed Location Service Architecture
7.2	Object Representation within an Office
7.3	Object Interaction with Searching
7.4	Handling Errors
7.5	Space Decomposition Using a Quad-tree
7.6	Space Decomposition Using an R-tree
7.7	Basic Architecture of the Location Service
7.8	Location Service Interface
7.9	Active Sides of Objects
7.10	Determining Facing Objects

7.11	Alternatives for Facing Area	70
7.12	Relationship of Indexed Object Records	73
7.13	Necessity for Clipping	74
7.14	Clipping in Practice	75
7.15	Use of Rectangles to Approximate a Convex Polygon Shaped Room	75
7.16	Austin Building, Floor 5	76
7.17	Floor 5 with Junctions	77
7.18	Searching using Junctions	78
8.1	Floor 1 of ORL	84
8.2	Floor 1 Containers	84
8.3	Floor 1 Junctions and Containers	85
8.4	Floor 1 Infra-red Zones	85
8.5	Error Derived from an Infra-Red Zone	86
8.6	Active Badge Interface	86
8.7	Error Associated with an Ultrasound Badge Sighting	87
8.8	Ultrasonic Badge Interface	87
8.9	Printer Locations on Floor 1	90
8.10	Equipment Capable of Accepting a Teleport Session	94
8.11	Effect of Node Size on Update Performance	96
8.12	Effect of Object Density on Update and Search Performance	98
8.13	Effect of Index Size on Update and Search Performance \ldots .	98
8.14	Search Performance	99
8.15	Location Service Update and Search Performance	101
8.16	Update Performance with Registered Search Entities	102
9.1	Evaluation of CE Expression	109
9.2	Types of Timer	112
9.3	Submission of CE Expression	l15
9.4	CE Monitoring Threads	116

List of Tables

5.1	Event Class Attributes	48
6.1	Illustra Update and Search Performance	59

Chapter 1

Introduction

Computing systems are becoming increasingly mobile, facilitated by advances in wireless networking, battery technology, the emergence of low-powered and low-cost portable devices, wider deployment of standard communications technology and changes in user working patterns. During the course of a day mobile users may interact with many different mobile and stationary computers in many different locations and situations. Interaction with computers is no longer restricted to a set of desk-bound devices. A new set of hardware types have emerged, such as networked multimedia peripherals, low-cost high-resolution displays able to be deployed anywhere on a network, and voice-controlled devices. Conventional office based systems such as telephony and environmental control systems are increasingly being integrated into computing infrastructures.

Due to these developments, a new paradigm of computing interaction has emerged: this is termed *context-aware*. A context-aware system is one in which applications have knowledge of their surrounding physical and computing environment. This environment is composed of people, mobile and fixed computing devices, and such things as doors, walls, desks and chairs.

This thesis proposes an architecture for supporting context-aware applications within a typical indoor environment such as an office or home. This architecture is called CALAIS, a *Context And Location Aware Information Service*.

1.1 Context-Awareness

Examples of context include the group of people a user is currently with, the types of equipment and communications technology available, and a user's physical location. For example, a mobile user may wish e-mail to be delivered automatically at the earliest possible opportunity, removing the need to return periodically to the personal desk-top to check for new mail. As the user moves, a context-aware e-mail delivery application needs to know the identity of equipment near the user and its suitability for delivering e-mail. Once an unused and suitable item of equipment is nearby, the user can be alerted and the e-mail

delivered. Another example is that of a user attempting to contact a colleague. A suitable context-aware application may determine that the colleague is in a meeting, implied by the fact that other users are present in the same room. The contact attempt may therefore be postponed until the colleague is alone. Once alone, the application may then determine whether video-phone equipment is deployed and unused near the colleague. If not the telephone may be used as an alternative communication device.

Context-awareness leads to:

• Automation

For example the automatic logging in of users to computer systems, and the redirection of telephone calls. Interaction with computing devices is simplified and made more casual.

• Adaption

An application may alter its behaviour depending upon the types of equipment available and the status of a user's surrounding environment.

• Personalisation

Users may specify the behaviour of an application within certain contexts. Ideally a user's personalised computing environment should be available wherever the user may be.

Context-awareness is a *user-centric* view of computing. The computing system dynamically and non-intrusively adapts itself to a user's circumstances, reducing the focus on individual computing devices. By the application of context-awareness a single device can exhibit different semantics in different contexts. This enables the interactive complexity of a device to be reduced, in turn reducing its physical size and power consumption. Context provides a powerful tool to prime users and equipment prior to interaction.

1.2 Location Information

Knowledge of the location of users and equipment is a prerequisite for the support of context-aware applications. Locating physical objects is itself a distinct research area which involves the development of location hardware devices, software storage structures, and mechanisms to enable location-based querying.

Location devices may give a location directly, such as in the case of Global Positioning System (GPS) receivers. A location device can also take the form of a tag, attached to a user or device which periodically communicates with a fixed receiver infrastructure. With knowledge of the positions of receivers, the location of the tagged object can be determined. Location may also be inferred by using, for example, electronic access-cards, or by monitoring the interactive use of equipment where the location of this equipment is already known. As far as the user is concerned, the gathering of location information should be as non-invasive as possible. Location information used in isolation is of only partial use. It may be used together with information about the domain in which it is gathered to enable its graphical visualisation or its presentation in a human readable form, for example "the surgeon is located in the operating theatre". Of particular relevance to context-aware applications is knowledge of the physical objects in the immediate vicinity of a *target* object, such as a person. Many applications are not interested in the absolute location of an object, but only its relative location to those objects nearby. This is termed *co-location*.

A mobile location device can not only give a location, but may also provide authentification of the identity of a user or device. Users conventionally authenticate themselves using a user-name and password. With authentification provided by location devices interaction can be enabled without user intervention and can thus be more casual, an important requirement for a mobile system.

1.3 Environmental Awareness

Location is only one example of environmental *sensing*. Many other aspects of a typical domain can also be monitored. Some examples are:

- *Networked device use:* the interactive use of a mobile or fixed computing device such as a workstation or telephone.
- *Cameras:* by examining video from cameras, movement can be sensed.

Doors: it can be determined whether doors are open or closed.

Light and heat levels: within each room, light and heat levels can be sensed.

The gathering of this information enables additional context to be derived from the environment. Many of the above will be sensed using hardware devices, some using software. Some context-aware applications will need to augment this sensory information with other, persistent information, perhaps stored in a database. For example, in order to determine whether a particular device can accept a video-phone session, the capabilities of equipment need to be known.

Applications may use this sensed information is a variety of ways. Some examples are:

- *Workstation usage:* if a workstation is already being used interactively, another nearby workstation is chosen automatically to enable the presentation of an application.
- *Telephone status:* an attempt at contacting a person by telephone is delayed until the telephone handset nearest to the person is replaced.
- Avoiding disturbance: a visit to an office is postponed because the occupant is using a video-phone.

• *Light level:* lights are turned on when the ambient light level is not sufficiently high.

1.4 Target Environment

In order to develop and evaluate the CALAIS architecture a specific indoor target environment is considered. This will provide requirements for aspects of the system and provide a test-bed for implementation.

The main target environment considered is that of the Olivetti and Oracle Research Laboratory (ORL) in Cambridge. To a lesser extent parts of the University of Cambridge Computer Laboratory are also considered. Both these domains consist of a number of floors, corridors, offices and meeting rooms. Both have a rich computing environment and are already equipped with a number of technologies suitable for the construction of a context-aware environment.

1.5 Research Aims

The research resulting in CALAIS will focus on the following main areas:

• Sensing technologies

A set of suitable sensor technologies are considered which will enable location and contextual information to be gathered from the target environment.

• Data delivery

Sensor systems produce widely varying types of information. General mechanisms are considered which aim to abstract sensor hardware and software. This will allow the delivery of sensor information within a distributed system and ensure simple addition of future sensor devices.

• Location information management

Location devices are considered, together with a canonical representation of a location. Requirements for the management of this information, and application query interfaces are examined. Furthermore, methods of representing the static environment, consisting of such things as walls and doors, are considered.

• Application support

Many context-aware applications will require information from multiple, distributed sensor systems and other resources. Consideration is given to methods of reducing in complexity the building of such applications.

4

1.6 Dissertation Outline

Chapter 2 reviews previous research into location and context-aware systems. This encompasses location sensing technologies, current approaches to the support of mobile applications, active database systems, methods of location management and asynchronous event systems.

Chapter 3 presents a critique of this related work and determines the requirements the CALAIS should fulfill.

Chapter 4 presents a method of abstracting sensor system technologies based upon an event paradigm. A set of deployed and prototype sensing technologies suitable for the monitoring of a dynamic environment are presented. These technologies provide the experimental data used in this thesis.

Chapter 5 proposes a method for building sensor driven services within a distributed environment. Further consideration is given to the event paradigm and an approach to building services using this model. Database support for persistent information is presented and pertinent issues relating to a distributed environment considered.

The design and implementation of a location data management system is presented in chapters 6 and 7. Issues considered include the integration of multiple location devices, ways of representing the location and orientation of physical objects, application query interfaces and performance requirements. A lightweight system is presented which aims to implement the requirements identified. An approach to the representation of the static spatial domain, such as rooms and doors is also considered. The functional capabilities and performance of this system are then evaluated in chapter 8.

Chapter 9 describes an approach to the creation of complex context-aware applications using multiple sources of sensor information. A set of implemented examples is described which aim to justify this approach.

Finally, chapter 10 presents a summary of this research and indicates areas of possible future work.

Chapter 2

Related Work

This chapter presents an critical overview of previous work pertinent to this thesis. Shortfalls in this work are collated in chapter 3 which then identifies a set of requirements for CALAIS.

2.1 Location Technologies

A context-aware environment requires knowledge of the locations of people and equipment. This section presents a review of some relevant technologies.

2.1.1 Active Badges

ORL have made a significant contribution to the area of location aware systems through the development of the Active Badge[WHFG92, HH94]. Active Badges are worn about the person and communicate a unique identifier approximately every 10 seconds using an infra-red transmission. Infra-red does not penetrate solid surfaces such as walls. Sensors placed at fixed positions within a building receive these infra-red signals and through software make the sightings available to services and applications.

Active Badges have a range of approximately five metres. Therefore one sensor per room is usually sufficient, resulting in the Active Badge system giving roomscale location. In large or irregularly shaped rooms the number of sensors can be increased to ensure full coverage. In these cases the granularity of location is potentially smaller than a room but is usually interpreted at the room-level.

Such room-scale location has proved to be very useful in working environments where cooperation with colleagues is necessary. One popular application displays a view of this data indicating the room a person is currently in and the person's nearest telephone extension. Such information provides not only methods of contact such as which office to walk to or which extension to telephone, but also gives hints about the person's present activity. For example, if a person is alone in an office a user may feel it is convenient to contact them. If, however, other people are also in that office, contact may be delayed until the person is alone.

Active Badges are also equipped with two buttons which, when pressed, immediately transmit an infra-red signal which additionally contains information indicating which button was pressed. This facility allows the badge to be used as a simple input device.

A variation of the Active Badge, the *equipment badge*, has also been designed to attach to items of equipment. It is assumed that equipment is less mobile than people and therefore these badges transmit approximately once every 10 minutes, extending battery life to around five years and therefore reducing system maintenance.

A number of Active Badge applications have been developed. The Active Map[War95] annotates graphical floor plans of ORL with Active Badge information from people and equipment. The topology of the office environment and location information has also been used to determine the nearest printer to a specified user. In addition, badges have been used as an authentification device to allow admittance to parts of a building and a car park. Another example of an application controllable with an Active Badge will be described in more detail in section 2.6.1.

Active Badges have been successfully deployed within a number of academic and industrial laboratories. Their small size makes them convenient to wear and to attach to devices. The main drawback is that only room-scale location is possible.

2.1.2 Electromagnetic Tracking

Electromagnetic tracking systems such as those produced by Ascension Technology[ATC97] are capable of resolving the location and orientation of a tracked object with very high spatial and temporal resolution. Magnetic fields are emitted by a single transmitter to track the position and orientation of a number of sensors. A location error of 1 millimetre is typical within an area of approximately four metres and up to 120 sightings per second can be obtained.

This type of technology is commonly used in the computer animation industry, where the movement of human subjects is determined by the attachment of such devices to various parts of the body. These systems are expensive and only operate within a constrained area of a few metres and are therefore not suitable for wide-scale deployment.

2.1.3 Optoelectronic Tracking

A system described in [Azu93] uses optical sensors built into a head mounted unit. These sensors are sensitive to infra-red beacons built into a room's ceiling. With knowledge of the positions of these beacons and which beacons are currently in view, the location and orientation of the user's head can be deter-

8

mined. The system is highly accurate, resolving locations within millimetres and orientation to within a fraction of a degree. However the system is costly to deploy and the head mounted unit large and unwieldy. It is not designed for wide-scale deployment or for tagging equipment.

2.1.4 GPS

Global Positioning System (GPS)[AL94] receivers use highly accurate time signals, transmitted by a number of satellites, to determine their position to within a few metres. By additionally using Doppler effects, the instantaneous velocity of the receiver can also be determined. However GPS transmissions are blocked by buildings and therefore cannot be used indoors.

2.2 Location Information Management

To deliver information from location devices to applications and to enable the maximum benefit to be gained from its gathering, software storage structures are considered necessary to provide higher-level management. Some of the work presented below has been directed towards context-aware system support, whilst other work considers more general hierarchical structures for use within wider-scale cellular systems.

2.2.1 Dataman

The Dataman project[IB94] at Rutgers University has investigated various aspects of mobile computing, and in particular data management issues arising in mobile, wireless environments. It is asserted that the mobility of users and services and its impact upon data replication and migration will be one of the main technical problems to be resolved. Although much of the work relates to data management within a wide-area cellular network, some of the issues that are explored are pertinent to context-aware systems. One of these relates to scaling, and the interest of remote clients in a domain's location data. A hierarchy of *location servers* is described, reproduced here in figure 2.1.

Users are assumed to be highly mobile and to move regularly between location server domains. It is suggested that each user will be permanently registered under one of the location servers. This will be known as their *home server*. Location updates will be propagated to the home server which will hold the authoritative database of locations. A client wishing to contact a user will always start by contacting their home server to find their current location.

Separately in [IB92] a strategy is described for the querying of data with a spatial constraint. It is assumed that the cost of providing a database with up-todate knowledge is too high and thus a database will have imprecise knowledge. Therefore queries which do not necessarily need the most up-to-date knowledge are satisfied, whereas in the case of queries which require greater accuracy,



Figure 2.1: Server Hierarchy

further demand paging of the user's home service is required.

This work suffers from the reliance placed upon the home server. As every change in the mobile user's location is propagated to the home server and some types of query also rely on the home server, reliability of this server as well as the network connections to it is essential. If either the server or network connection fails, the location of the mobile which it supports will be temporarily unavailable.

In a later Dataman paper, Welling[WB97] recognises the need for applications to be made aware of changes in the networked computing environment, such as the connection of a hardware component or the running down of a battery. An architecture is proposed which allows applications to be notified of such environmental changes. To illustrate the benefits of such a system the *Pine* mailreader was modified. This version of Pine is intended to run on laptops which have a cellular modem link in addition to the ability to connect to a tethered network. The system produces an event when the computer is connected or disconnected from the tethered network. Pine receives this event and changes its mail dispatch behaviour by batching mail messages when untethered and sending messages immediately when tethered. This illustrates an application reacting to a change in context.

2.2.2 Hierarchical Location Management

Krishna et al[KVP94] also describe a hierarchical system for location information management. A hierarchical tree based scheme is used, similar to that used in Dataman. Leaf nodes represent base stations and internal nodes represent location servers. In Krishna's scheme each location server maintains information regarding mobile hosts residing in the subtree beneath it. Updates and searches are supported. Each location server keeps a table of 3-tuples. These consist of:

• A mobile host identifier. This is unique and also provides the address of the host's home location.

10

- A forwarding pointer. This identifies which location server the host has moved to.
- A time-stamp. The time the last forwarding took place.

Base stations maintain a similar structure for each host contained within its cell. By using these forwarding pointers various update strategies can be used, varying in expense and efficacy. Periodically the forwarding pointers are collapsed and a single pointer created. Searching is conducted by progressively moving up the tree until a location server is found which contains a record for the required host.

Host behaviour, expressed in terms of communication frequency and mobility, is not normally known to the designer of a location server hierarchy. Thus a set of *dynamic* extensions are made to the model which assume that the past history of the system will reflect future behaviour. The basic idea is to alter the search and update strategies based upon the mobility profile of the host. Such an approach is shown to lead to a more efficient model and implementation.

2.2.3 A Location Service Design

Leonhardt[Leo95] states that the provision of location information is an important resource for mobile systems. A system providing such information must minimally provide facilities to:

- find the location of an object;
- find all the objects at a given location.

Leonhardt goes on to describe a hierarchical model consisting of location domains containing a set of overlapping location zones which in turn contain a set of cells. The size of these cells determines the granularity of the location system. It is asserted that such an architecture allows the integration of multiple sources of location information. Location domains are themselves arranged in a hierarchy to allow hand-over and roaming.

Although this architecture appears promising, little detail is provided and the implementation is undeveloped. Evaluation of this work's worth is therefore difficult.

2.2.4 Integration of Location Hints

Rizzo et al[RLU94] comment that different location systems such as the Active Badge, system login information and personal location diaries are normally used in isolation. A location system is proposed which attempts to combine information from as many sources as possible, forming a *master location system* (MLS), controlling a set of slave systems. Two main issues are identified relating to how information from different sources should be combined:

- Uncertainty. Any location system will necessarily give uncertain output, largely due to the fact that sightings are reported in a discrete, not a continuous manner. Active Badges transmit every 10 seconds and their transmissions can be obscured, for example, by clothing. If the last sighting reported is 1 second ago, this is assumed more valuable than a sighting 30 seconds ago, by which time the user may have left the room. A confidence value is associated with each reported location. This depends upon the reliability of the technology involved and the habits of the user.
- Corroboration and conflict. Two slave location systems may return the same, or a different location for the same user. Confidence values are used by the MLS to resolve these potential conflicts. Suggestions are made for a corroboration function, although it is unclear how these would work in practice.

The model requires the correspondence between different location systems to be made explicit, for example that a particular workstation is contained in a particular room. These relationships are encoded allowing the MLS to correctly resolve sightings from various location servers.

2.2.5 Active Maps

Schilit describes the *active map server* (AMS)[ST94] which publishes the positions of tracked physical objects. Each AMS supports a set of spatial containers arranged in a hierarchy. This hierarchy might represent a building and the rooms contained within it. Schilit suggests that each AMS should serve a well-defined geographic area, such as a building, thereby taking advantage of user 'locality of reference'. User movements will be common within an AMS domain and less common between domains.

Location is supported in terms of containment, provided by Active Badges and cell-based radio, both which provide room-scale location. Distance between containers is supported by the construction of a graph enabling the shortest path algorithm to be applied. The distance between two physical objects in different containers can then be approximately determined. Applications can subscribe to an AMS and receive notification when objects of a particular type move. A one-off query interface is available to execute a container based query.

The nature of the AMS allows the objects in any container within its hierarchy to be determined. However no discussion is presented of software storage structures which will allow such queries to be executed efficiently. Also, no indications are made of the performance of the system and how many objects can be supported. Finally, no provision is made for the support of fine-grain location information.

2.2.6 GIS

A Geographic Information System (GIS)[Wor95] is an information system enabling the representation, analysis and visualisation of geographically referenced data. A GIS works with large scale spatial areas, such as a city or country. It allows the representation of geographic entities such as terrain contours, rivers, forests, roads and houses. Analysis can then be conducted using this information. Examples of queries are:

- Find the shortest road route between towns A and B.
- Find all locations on a sand deposit and within 500 metres of a major road.
- Find regions of forest felled in the last five years.

The last example involves a temporal aspect and requires the GIS to timestamp information and be able to execute temporal queries. This recognises that geographical information is not static but changes regularly.

A GIS will typically use a database management system at its core. This system may be general purpose, for example *Postgres*[vOV91], or more specialised, for example *Smallworld*[The96]. The system used must support spatial types to represent geographic entities and methods of indexing these spatial types so allowing fast access to the data held.

Although both deal with spatially referenced data, a GIS differs in a number of ways from a location management system suitable for use within a mobile environment. One major difference is that data stored within a GIS typically has a long lifetime, perhaps only changing over a period of years. Secondly, although fast querying facilities are desirable for interactive use, it is not necessary that a GIS responds in real-time, as opposed to a mobile system where real-time notification of location changes is essential.

2.3 Environmental Sensing

The gathering of location information is an important example of environmental sensing. This section describes some approaches to monitoring other aspects of the physical environment.

2.3.1 The Interactive Office

The aim of the *interactive office*[HL94] is to provide information about a user's context by the use of a range of sensors embedded within the physical environment. The primary motivation cited is to prevent unwelcome interruptions, for example, a telephone call or a knock at an office door when in a meeting. By publishing information about a user's context it is envisaged colleagues will be able to interact with each other more sensitively.

Prototype hardware enables the system to detect whether doors are closed, whether chairs are in use, the sound level in a room, movement through doorways and fine grain movement around, for example, a desk.

This information is gathered and evaluated in software using a rule-based system which enables pre-defined actions to be triggered. An example application area is telephone system control. If, for example, a meeting is taking place in a particular office and a call is put through to that office, the telephone only rings once although the connection persists until the caller terminates it. The occupants are thus alerted with the minimum of disruption and can either answer the call or ignore it.

2.3.2 Monitoring Computer Use

With knowledge of a computer's location and the identity of the user of that computer, the location of the user can be estimated. In [ST93] and [Leo95] a set of networked Unix workstations are monitored using the *rusers* daemon, which publishes the user-ids of the people currently logged-in to a machine. It is unclear however how applications may gain access to this information, nor how this information may be used.

2.3.3 An Application Interface

Schilit[Sch95] tackles the problem of communicating changes in the software and physical environment to Unix applications. Consider a Unix shell. Defined within the shell are a set of environment variables, containing information such as the name of a default printer. These variables are typically defined once, and their values persist until the shell terminates. Applications invoked by the shell will also use these variables, often reading their values on startup. A 'printer' variable, for example, will normally hold the name of the nearest printer and have its value set at application invocation. If an application and user are mobile and move location, the value of the printer variable will no longer be appropriate.

A dynamic environment server manages a set of objects, similar to traditional environment variables. An application can register an interest in changes to a managed object. When the object's value changes the application is informed. A system may employ any number of environment servers which will publish information from sensors and other data sources. Applications can therefore re-configure themselves dynamically as the system resources in their vicinity change.

2.3.4 SPIRIT

The SPIRIT (SPatially Indexed Resource Identification and Tracking)[ASH97] project at ORL aims to build an authoritative resource database holding information about many aspects of the networked computing environment. The

database represents entities such as workstations, Active Badges, telephones, people and network peripherals. Most entities have both static and dynamic information associated with them. Static information, such as a name, will change infrequently, whereas dynamic information, such as the location of an Active Badge, will change often. The information associated with a workstation, for example, includes processor type and speed, the type of network interface, spare disk capacity and CPU load.

Clients of the SPIRIT database may include a load balancer requesting CPU load information, a visualisation application querying Active Badge information and a system administrator requesting a list of equipment of a particular type.

An Oracle V7 database management system is used as the primary data store. This provides facilities such as backup and recovery mechanisms, query optimisation and indexing. A declarative language allows the production of a database schema and a remote procedure call interface. The latter allows an application to use the SPIRIT system without having knowledge of SQL.

A critisism of the SPIRIT project is its reliance on a centralised DBMS. Although the information sources are distributed over a local area network, all information is propagated to the central database. This leads to scalability problems when the amount of monitoring or the number of applications using the system increases. Also, a heavyweight DBMS is not necessarily the most suitable technology for handling rapidly changing data, due to the overheads necessary in ensuring the consistency and durability of each update.

2.4 Ubiquitous Computing

Ubiquitous computing is a paradigm first described by Weiser[Wei93]. The motivating critique is that current computing systems are computer-centric, that the computer itself becomes the focus of work, rather than simply a tool to aid work. The goal in ubiquitous computing is to augment the everyday environment with computers that will integrate seamlessly with a person's habits and work patterns. Weiser considers that ubiquitous computing is the opposite of virtual reality in that in does not attempt to replace the normal world, but rather to integrate with it.

An initial prototype system is described, consisting of three types of computing device: tabs, pads, and boards. A tab can be compared to a Post-it note. It can be held in the palm of the hand and consists of a small pressure sensitive screen and an infra-red transceiver. Pads are larger, are controlled with a pen and have a $64Kbit \ s^{-1}$ wireless link. Boards mount on walls and connect to a wired network. A board can can be considered the electronic equivalent of a whiteboard.

An example of an application using these devices [EHC⁺93] explores a ubiquitous computing approach to energy management and environmental control within the PARC lab. A tab can, using infra-red information, discover which room it is currently in. A person can then use the tab to change various environmental settings within that room, such as temperature, humidity and light levels.

2.5 Augmented Reality

Within an *augmented reality* environment the user is typically equipped with one or more devices, possibly worn about the person. An example is a headmounted display. The user moves within the real world and virtual reality techniques are used to augment the user's conventional view with additional context-aware information.

2.5.1 Knowledge Based Generation

Work by Feiner[FMS93] concentrates on knowledge-based generation of threedimensional virtual worlds that complement a user's view of the real world. The system uses an illustration system in association with a see-through headmounted display. An example application task is that of printer maintenance. The printer is equipped with sensors which inform the system about the printer's physical state, for example whether the paper tray is open. This information is then used to decide whether to project images of the printer onto the headmounted display. If the user is looking at the printer, and the printer tray is not open, the system will project an image of the open paper-tray, thus indicating its function and how to gain access.

2.5.2 Wearable Computing

Starner et al[SMR⁺97] describe *wearable computers* which use head-up displays to overlay graphics, text and sound onto the wearer's normal view of the physical world. In one example, the editor *emacs* is projected using the head-up display so the image overlays the user's view of the real world. Pull-down menu options are displayed as usual along the top. By mounting a video-camera on the user's head which transmits its output by radio for remote processing, the user's finger is recognised and its position interpreted in the same way as the position of a mouse pointer on a conventional display. The user can therefore select menu options and carry out other mouse-oriented functions.

Another example describes a museum scenario. Again, the user wears a headup display and a camera. Physical markers encoding a unique binary pattern are placed upon objects of interest. When a marker is recognised, the text, graphics or video behind the hyper-link is projected onto the head-up display.

2.5.3 Situated Information Spaces

A variation on augmented reality are *situated information spaces*[Fit93], which use hand-held rather than head-up displays. The idea is that a user equipped
with a tracked palm-top computer will receive information based upon their proximity to objects within the real world. For example a user may 'browse' the space near a fax machine. By browsing the in-tray, information about received faxes will be displayed on the palm-top. Another example is that of interaction with paper-based displays such as maps. The palm-top acts as a window on to the map by displaying relevant information about the map area the the palm-top is currently positioned near. A zoom control on the palm-top allows access to more detailed information. Such a system could be used in a library, where electronic information would help guide users to the written publication they require. The tracking unit used is a 6D device, giving x, y, z positional coordinates as well as the pitch, yaw and roll orientation of the palm-top device. However this device can only work within a 1 metre cube, thereby severely limiting its application.

2.6 Context-Aware Applications

2.6.1 Teleporting

Teleporting[BHR94] is a method of making an X-windows user interface mobile by enabling it to relocate between displays. In conventional X systems, once an X client application has started and initially connected to an X server, it cannot break this connection and reconnect to another server. Relocation requires the termination and restarting of an application.

Teleporting addresses this problem by providing a *proxy* X server which for each user relays communication between clients and real servers. When display relocation is required, the proxy server manages the disconnection and reconnection to real servers. Clients are unaware of this change. A teleport X session can be *materialised* and *de-materialised*, to and from a display, using simple commands.

Teleporting has proven valuable within the environment of ORL, where collaborative work is common. A familiar scenario is where a user visits a colleague's office to discuss a problem. The user may materialise their teleport session on their colleague's display so enabling them to explain the problem more conveniently.

A teleport session can be controlled with the buttons of an Active Badge. A press of the side button informs the proxy server that the user wishes to materialise their teleport session. By the use of equipment badges indicating the location of equipment, the proxy server can find a list of nearby displays, which then uses simple heuristics to select a display on which to materialise. Such control illustrates well the importance of knowledge of co-located objects at a user's location.

A recent derivative of teleporting is described by $[WRB^+97]$ and termed the *virtual network computer* (VNC). The motivation for this work is to enhance the power of teleporting by reducing the dependence upon access to an X equipped

machine. Advantage is taken of the proliferation of the World Wide Web and Java equipped browsers, so making the teleporting system globally available using the Internet. A Java applet provides an interface to the user's proxy server at their home domain, rendering X clients on the browser, and passing on mouse and keyboard interaction to the proxy server.

2.6.2 Mobile Agents

Bacon et al[BBH97] describe a framework for building location-oriented multimedia applications. In contrast to teleporting, where an application's user interface moves, *mobile agents* use application code and execution state which can move from one host to another. This differs from process migration systems in that no operating system support is required, allowing system independence. An interpreted language is used. Execution of this can be halted, its state saved, and the code and state moved to another machine. Execution can then restart. Location information, from Active Badges, is available. By monitoring the location of a user, and also having knowledge of the equipment available nearby, running applications can move in response to user mobility. Multimedia applications have provided a testbed for these techniques. A mobile agent manages the application and when a user moves the agent reconnects the application to new multimedia endpoints, such as cameras and displays.

2.6.3 Stick-e Documents

Brown[Bro96] describes a framework for creating context-aware applications. A *stick-e document* is composed of a set of *stick-e notes*, each resembling a page of HTML. Each stick-e note consists of content, and the context in which it will be triggered. Consider a mobile user carrying a PDA equipped with location sensing hardware. The user can place a stick-e note at a physical point of interest. When the user returns to that position in the future, the stick-e note will be triggered, the user being informed of this by the PDA. A stick-e note can therefore act as the electronic equivalent of a Post-it note. Some other examples of context that could trigger a stick-e note are given: the adjacency of a person to other physical objects and when the temperature is below a certain level.

The stick-e note approach offers a useful general mechanism for the creation of context-aware applications. Further work appears to be required on system aspects, that is the management of location information and the delivery of contextual information to applications.

2.6.4 Memory Protheses

Lamming et al[LBC⁺94] contend that the execution of everyday tasks, such as finding files, recalling events at a meeting and the remembering of appointments, can be eased by the use of a set of applications they term *memory protheses*.

Electronic organisers and personal digital assistants are critisised as being little more than sophisticated notepads. A memory prothesis, on the other hand, will be sensitive to a person's surroundings and actions, and will silently record information about context enabling recall at a later time.

Pepys[NEL91] is a precursor of the memory protheses work. Pepys involves recording the movements of people around the Xerox-EuroPARC laboratory using Active Badges. By recording the time spent in a place and the people also present, a movement diary can be built up. In use, such diaries prove useful in enabling people to recall events such as "what people were at the meeting at 10.15am on July 19th?"

More recently other types of data have been captured, such as video footage, handwritten notes and document use, both paper and electronic. Lamming et al identify a set of system requirements for protheses support. These include:

- *sensing of the environment:* location information, whether a door is open or closed, telephone activity etc.
- *automatic data capture:* as much data should be captured as possible as it is not known *a priori* which items of data are important.
- *manual annotation:* where appropriate a user should be able to enter an explicit record.
- *privacy:* traditional diaries are regarded as confidential. A prothesis should be treated in a similar way.

2.6.5 Computing Personae

Banerji et al[BCK93] describe the concept of *mobile computing personae*. The idea is that a user will have a single computing persona, which will present itself in different ways, depending upon the equipment that the user is currently working with. For example the user may specify that an e-mail client is always displayed. The type of e-mail client, whether graphical or text-based, will be chosen by the persona to match the equipment available.

Such a concept has many implications, for example, a particular application operating within a persona may require a co-located set of computing devices of certain types. Also, for a persona to be compatible across platforms, information about equipment and software capabilities needs to be available. Banerji describes a *resource representative* which enables clients to find out the capabilities of equipment. Equipment exports information about itself to the representative such as operating system type, memory size and communication connections, and clients may then query the representative. A *persona manager* is responsible for creating and maintaining personae. A persona can be seen as a *view* of resources, a way of abstracting machine specific applications.

2.7 Event Management and Composition

Changes within computing systems can be modelled as *events*: atomic, asynchronous occurrences at distinct points in time. Examples of events include a hardware peripheral interrupting a processor, an update to a database, and an Active Badge sighting.

Much research effort has been focused on the role of events within active database systems [GJS92, CM93, WC96]. This has been within the context of the *event-condition-action* (ECA) rules used in such databases. The action of an ECA rule is triggered when the event is detected and the appropriate condition satisfied. Chakravarthy[CKAK94] states that methods of event specification have applicability in any application area where asynchronous occurrences are required to trigger an action.

A system will typically have many sources of events. A client may be interested when a particular sequence of these events occurs. For example "tell me when user *Giles* is seen in room 501 and he is active at workstation *scallop*." Such event *composition* could be left for applications to deal with in an ad-hoc way. Research has however been conducted into how best to generally provide such facilities. Various languages for expressing composite event semantics have been proposed such as those presented in [GJS92, Bat94, BBHM95, Hay96]. Bacon et al[BBHM95] also incorporate concurrent monitoring of composite events, as well as noting the importance of time.

2.7.1 Distributed Events

Within a distributed software environment the sources of events are usually termed *services*. A client application may then contact the service and request to be informed of all events that the service is responsible for.

Schwiderski[Sch96] explores aspects of event service distribution and the problems this causes. For example, each machine within a distributed system has its own clock. If these clocks are not synchronised then difficulties arise in evaluating time-based composite event expressions where the constitute events originate from different source machines. Network delays may also result in the incorrect temporal delivery of events to a client.

2.8 Review

This chapter has given an overview of related work relevant to context-aware systems.

Devices used to gather location information (section 2.1) are varied in nature, producing many different types of location data. The importance of location information within a context-aware environment has been demonstrated by the development of the Active Badge and resulting applications (sections 2.1.1, 2.6.1, 2.6.2, 2.6.4).

Approaches to the management of location information (section 2.2) were described. These are currently simple and are typically designed for a particular location technology. Support for multiple technologies, and different granularities of location information has not been considered.

Attempts have been made to monitor the dynamic environment (section 2.3). The resulting information has been used in a variety of application areas (sections 2.4, 2.5) together with location data.

A potential data delivery mechanism exists in the form of an asynchronous event model (section 2.7). Work has also been described which aims to support applications requiring information from multiple distributed sources.

Chapter 3

Analysis and Requirements

This chapter provides a review and critique of the related work presented in chapter 2. Following this, a set of requirements for CALAIS are identified.

3.1 Review and Critique of Related Work

3.1.1 Applications

Applications provide the motivation for context-aware and location system support. A number of application areas have been identified. The functionality of these can be summarised as follows:

• Locating tracked objects

A user finds the location of a particular physical object in order to interact with it, for example, the location of the nearest workstation. Tracking objects enables graphical visualisation of an environment, perhaps updated as objects move.

• Spatial querying

An application finds the objects within a particular spatial region, for example a room. Alternatively it may find the nearest object of a particular type to a specified location.

• Automated application control

Applications and user interfaces are automatically triggered and moved from one device to another, based upon the location of a user and the capabilities of equipment. Environmental sensing information is also used to trigger applications, such as the re-routing of telephone calls.

• Augmented reality

Information from a wide range of sensors and location devices is used together with visualisation technology to augment the user's view of the real world, for example the printer maintenance example described in section 2.5. Sensor information comes from the user and from the surrounding environment.

3.1.2 Sensor Systems

A context-aware system needs to gather information about the user's environment. A set of embedded sensor technologies and location tracking devices have been developed to facilitate this. These include systems to monitor such things as chair use, door status, temperature, light levels, movement through doorways and user location. Whilst the devices which provide this information are interesting, little consideration has been made to how information from these systems is delivered to applications.

Sensor systems have typically been built in an ad-hoc way with the interface defined by the developer. The application writer therefore requires detailed knowledge of a number of technologies and applications become complex and non-extensible.

3.1.3 Location Information

Software approaches for managing location information within a context-aware system have typically centred upon a particular technology, such as Active Badges. Other approaches to location information management have concerned themselves with wider area cell-based networks and GIS systems. Cell-based systems are mainly concerned with migration from one cell to another and how a location system can be managed in a hierarchy. In general, locationbased querying is not supported. GIS systems in contrast, are concerned with directly supporting spatial information and allowing sophisticated querying to be executed. However this sophistication leads to high complexity and poor interactive performance.

Drawbacks with current approaches can be summarised as follows:

• Restrictive view of location information

Typically a container (room) based view is taken of location information, often being motivated by the use of Active Badge technology. This presents problems when other types of location technology are used which produce finer-grain location information.

• Simple physical object representation

Physical objects are typically represented as being located within some space. There is no notion of the size of spatial extent of the object itself, that is its length, width and height. With container based technology this view presents few problems, but it is not sufficient when dealing with finergrain location systems. The representation of the orientation of physical objects is also an unexplored issue. This is required to enable some types of location based queries to be evaluated, such as *"is the user facing the workstation screen?"*

• Rudimentary application interfaces

Current systems support the querying of a particular container-based space, discovering the names of objects within that container. Typically, it is also possible to find the location of a particular object, the query returning the name of the container the object is within.

Applications typically have to poll the location management system to be informed of updates. Asynchronous call-back mechanisms have been developed but these are system specific and only apply to the movement of objects, rather than movement within specified spatial areas.

• Static environmental representation

Little use has been made of information encoding the static physical environment, such as walls and doors. It has only been used as an aid to visualisation and has not been used to increase the sophistication of querying facilities.

• Performance

Context-aware applications often have to respond in real-time. This imposes restrictions on the design and complexity of applications and supporting systems. Current designs appear to have taken no account of performance.

3.1.4 Client Interfaces

Current approaches require applications to have knowledge of individual sensor system interfaces, supported by a number of operating systems. Little investigation has been made into how to ease the building of applications which require access to multiple event sources. The semantics of such *composite event* applications are currently hard to define.

Methodologies exist for the handling of composite events, originally developed within database systems. This work has been generalised to allow the handling of multiple events within distributed systems. These approaches potentially offer semantic richness whilst allowing applications to be built easily.

3.2 CALAIS Requirements

From a consideration of the problems with current approaches to context-aware systems, a set of requirements for CALAIS can be identified.

3.2.1 Sensor System Interfaces

Applications rely on receiving information from a variety of sensor systems, constructed using a range of hardware and software. A standard approach should be applied to the construction of interfaces to these sensor systems and a single paradigm of data delivery used, enabling accessibility within a distributed system. This will simplify the task of application development and make the system extensible, allowing the simple integration of future sensor system technologies.

3.2.2 Location Information Management

Location information will be produced from one or more sensor systems. To gain full advantage of this information, higher level management is required to enable the execution of spatial-based queries. A single model of location should be used, enabling the use of many types of location device within a single system. This model should allow the representation of physical objects themselves and their relative orientations.

Static environmental information, such as that representing walls, floor and doors, should also be integrated. This will add to the semantic richness of the location model and will allow account of these human dividing spaces to be made in the evaluation of location based queries.

Applications should be able to monitor not just the movements of named objects, but also to be able to specify interest in regions of space. Activity within these spaces should be communicated to the application as it occurs, without the need for the application to repeatedly re-submit a query. Application access to these facilities should be similar to that provided to sensor systems.

3.2.3 Provision of Persistent Resources

A context-aware system is highly dynamic. However, some examples of persistent resources required are:

- Sensor system naming and typing information.
- Environmental information encoding static aspects of the physical environment.
- Physical object names, dimensions and capabilities. A context-aware application may choose to use an item of equipment based upon one of its capabilities, such as whether it can deliver audio.

3.2.4 Client Facilities

Applications require naming facilities in order to initiate contact with sensor and persistent services. Also, the use of a distributed system raises the issues of distributed time and network delays, which must be resolved on a systemwide basis if applications are to receive correctly ordered changes in the sensed environment.

As an application writer, the building of an application using information from multiple sensor systems, persistent resources and location management systems, is inherently complex. Constructing an application should be simplified, enabling full functionality to be preserved, but hiding the complexities of the underlying system. This requires the provision of client libraries which enable access to the full range of CALAIS resources.

3.2.5 Performance

Context-aware applications will often operate with real-time constraints. An application may be made ineffective if it does not react to changes in context within a short time period, typically dependent upon levels of mobility. Performance must therefore be considered in the construction of CALAIS components.

3.3 Chapter Summary

A set of requirements for the CALAIS architecture have been identified. Figure 3.1 gives an overview of these components, built around a distributed system.



Figure 3.1: CALAIS Architecture Overview

Chapter 4

Sensor Systems

The efficacy of a location and context-aware system is dependent upon the quality of information gathered from monitoring the environment. The sources of this information can be generally termed *sensors*, which periodically report changes in the physical and computing environments.

This chapter will describe a set of deployed and prototype sensor systems which provide the experimental data for this thesis. As will be shown, the sensor systems use a wide-range of hardware and software, and the interfaces to each is ad-hoc in nature. Chapter 5 presents an approach for the abstraction of these interfaces and details the available output from each sensor system.

4.1 Active Badge System

The functionality of the Active Badge system [WHFG92, WH92, HH94] was briefly described in chapter 2. Here the system will be described in more detail.

The system consists of a set of badges worn by users, a set of sensors permanently fixed to walls, and a software architecture enabling the gathering of sightings and the dissemination of these to applications.

Badges and sensors communicate by using infra-red (IR) with a data rate of 9600 baud. These two features allow transceivers to be small, cheap and low-powered. The frequency of infra-red used does not pass through, but reflects off, walls and doors. Each receiver defines an *infra-red zone*, the size of which is dependent upon the sensitivity of receivers and the power output of badge transmissions. Receivers are wall-mounted and an IR-zone approximates to a hemisphere five metres in diameter. Most rooms can be represented with a single IR-zone. This results in the Active Badge system generally giving room-scale location.

Active Badges

Active Badges are worn about the person and transmit approximately every 10 seconds. Power consumption is reduced by the use of a light dependent resistor which increases the transmission interval when the badge is in darkness. Badges can operate for up to one year using a single set of batteries.

The badge is equipped with two buttons, two LEDs and a small loudspeaker. When a button is pressed, the badge transmits immediately with additional information specifying which button was pressed. The badge can also receive signals during a small time window after each transmission. The transmission received may activate the loudspeaker or illuminate an LED. By using these facilities the badge can act as a simple control and paging device.

Badges are also equipped with a radio sensor and are designed to transmit immediately when a a radio field is encountered. Low-powered radio transmitters which transmit a pulse-width modulated signal using one of a set of widths can be placed around doors and desks. The badge transmits additional information in its payload identifying which one of the set of radio frequency intervals it has detected. By using this information and using colouring to distribute radio transmitters, finer grain location can be obtained.

Equipment Badges are designed for less mobile objects, typically equipment. These are physically smaller and transmit only. The infra-red signal is only transmitted approximately once every five minutes. Typical battery life is five years thereby reducing maintenance overhead. The equipment also has sockets enabling two digital inputs to be monitored. The input values are encoded within the periodic transmission.

Sensor Network

Sensors are distributed at fixed positions throughout a building, typically mounted on walls. Normally a single sensor is used per office but more may be used if the office is large, or the room topology would prevent a badge transmission being received by a single sensor. Figure 4.1 shows an arrangement of sensors and how these are connected to a general purpose computer network.

Sensors are connected directly to a simple wired network which also provides a power supply. This enables ease of installation in inaccessible places. The sensor network is then interfaced to a general purpose computer network so allowing ease of data access.

Software System

Existing system software provides application interfaces, naming services and the ability to exchange information with other sites equipped with Active Badge systems. The software is implemented using the Advanced Network Systems Architecture (ANSA) Testbench[APM90], a platform for building distributed systems. A *naming* service allows a textual name to be associated with badge



Figure 4.1: Layout of Badge Sensor Network

and sensor identifiers enabling clients to provide more readable user interfaces. An *exchange server* enables information to be exchanged between sites, for example between the Computer Laboratory (CL) and ORL. Time-stamps are applied to badge sightings as they are received from the sensor network. Clients gain access to badge information by registering with the badge server software. Sightings are then communicated to the client asynchronously using a remote procedure call (RPC).

4.1.1 Deployment

The Active Badge system has been deployed at several academic and industrial sites. It is also available as a commercial product.

Badge information used in the context of this thesis comes from two sites, ORL and the CL. Sensors are deployed throughout both sites. Approximately 60 people and 150 items of equipment are equipped with badges.

4.2 Ultrasound Badge System

The Active Badge system has demonstrated that an office based location device, suitable for tracking people and equipment, is useful. Research has been recently conducted at ORL leading to the development of a finer-grain location system using ultrasound[WJH97].

By measuring the time at which a single source of sound reaches a known fixed point, multilateration can be used to determine accurately the position of the source.



Components of the ultrasound badge system are shown in figure 4.2.

Figure 4.2: Ultrasound Badge System

This consists of a set of ultrasound receivers fixed on the ceiling of a room, and a radio transmitter. Each physical object to be tracked is equipped with a badge, consisting of an ultrasound transmitter and a radio receiver. Several times per second a signal is transmitted on the radio channel indicating which badge should transmit an ultrasound signal. All badges decode this signal and the badge addressed transmits an ultrasound pulse. This is then received by the ceiling receivers. Software is used to calculate the distance of the badge from each receiver and therefore the badge position. The output from this system is a three-dimensional position accurate to within approximately 0.1 metres.

Typically each badge will be told to transmit on a round-robin basis. However the badge of a highly mobile object can be 'targetted' by the system and told to transmit more frequently. By using this facility the relative mobility of objects could be established beforehand, or be calculated dynamically by determining the rate of movement of each object.

4.2.1 Deployment

The ultrasonic badge system has been deployed in one room at ORL.

4.3 Workstation Activity Monitoring

People working in a computer-rich organisation spend significant amounts of time working with a computer directly. Such use can be monitored and the following information determined:

- when a workstation is busy;
- the identity of a person using the workstation;
- the approximate location of a person using the workstation if the location of the workstation is known (or vice versa).

Activity implies context. If a person is active at a machine it may imply they are busy. The machine may also provide a conduit for communication, such as by video-phone or e-mail. The method of contact will depend upon the capabilities and network connectivity available to the machine. In addition, activity monitoring can also be used to control applications. For example, an application may begin to monitor a workstation when a user's teleport (section 2.6.1) session materialises. When monitoring has determined that the workstation has been idle for a time, the teleport session may be de-materialised.

Workstation activity information has been gathered experimentally in previous work[Leo95, Sch95]. These systems have used the Unix *rusers* system. This enables the users who have a login session on a machine to be determined. However this is unsuited to a distributed environment as a user can login to a machine remotely. Furthermore the system proposed here provides direct support for ORL's teleporting system.

4.3.1 Monitoring X-Windows Activity

The approach taken here recognises that X-Windows[Nye92] is the standard workstation user interface used both at ORL and the CL. In addition teleporting is widely used.

The *xmonitor* daemon monitors a given X-Windows equipped workstation for mouse and keyboard activity. Periodically this activity is reported to an xmonitor marshaller which receives data from many xmonitor processes, providing applications with a central point of access to xmonitor information. This is illustrated in figure 4.3. Once xmonitor has sent an activity notification to the marshaller it waits a time (default 10 seconds) before monitoring again. This prevents applications being overloaded with activity information when, for example, a user is typing fast.

The user identifier of the person using the machine can also be determined. Normally this will be the user-id of the person logged in to the machine itself. However, other users may teleport to a machine. xmonitor recognises when teleporting occurs and correctly reports activity notifications as originating from the owner of the teleport session.



Figure 4.3: xmonitor and Marshaller Structure

4.3.2 How xmonitor Works

Each X-Windows equipped workstation runs an instance of xmonitor. At boot time machines start xdm, an X client which allows users to login. xmonitor is started after xdm, allowing it to connect to the root X session and begin monitoring immediately even if no person is actually logged in.

Xmonitor monitors for three types of X event: those produced when windows are created, when the mouse is moved in or out of a window, and when a key is pressed. Keyboard and mouse events are bound to a particular X window. By recording the ownership of windows, xmonitor can determine whether a keyboard or mouse event originated from the workstation's root X session or from a user's teleport session. In this way the correct user-id can be attached to the activity notifications.

4.3.3 Deployment

xmonitor has been deployed on X-Windows capable Unix workstations throughout ORL. This results in a total of approximately 25 machines monitoring a user-base of 50 people.

4.4 Door Position

Doors are a convenient way of making a room accessible and secure. Whether a door is open or closed may be of interest to a security application, or one ensuring fire-door regulations are met. The position of a door may also imply context about the occupant within an office. If fully open, the individual may be indicating that visitors are welcome. If ajar, the occupant may appreciate a knock and pause before a visitor enters. If shut, the occupant may be implying they are busy and do not wish to be disturbed.

4.4.1 Providing Door Status Information

A magnetic reed switch is placed on the door frame with a magnet placed at the corresponding position on the door. With this equipment it can be determined whether the door is open, or closed. When the door is shut the reed switch will be closed and when the door is open, the reed switch will be open.

The reed switch is connected to the parallel port of a PC running Windows NT which in turn is connected to an Ethernet network. Using an appropriate device driver the status of the reed switch can be determined using software. This can then be passed on to applications. This process is illustrated in figure 4.4.



Figure 4.4: Door Status Service Structure

4.4.2 Deployment

The door of room Au501 in the CL has been equipped with a reed switch and magnet.

4.5 Telephone Handsets

Telephones can form both a source of information and an application area. *Computer Telephony Integration* (CTI)[Wal96] is an emerging and broad application domain which attempts to integrate computer communications with telephone networks. Amongst others, some basic aims of CTI are:

Screen based telephony: using a computer to make and monitor telephone calls.

- Call based data selection: using telephone call information to display appropriate data relevant to the caller and callee pair.
- *Data transfer:* transferring data from a telephone to a computer application and vice versa.

At present, the telephone-PBX in use at ORL does not allow the monitoring of the system state, nor the control of telephony devices by computer. To consider the impact of a limited form of CTI, magnetic reed switches are placed in the bodies of ordinary telephones. Magnets are placed in the handsets. The reed switches are connected to a Windows NT PC which makes this information available to applications in a similar way to the door sensor described in section 4.4. When the reed-switch is closed the handset is present on the base. When the switch is open this indicates the handset is raised, thus implying the telephone is in use.

Using this information, an application can determine whether or not to initiate contact with a user. If the user's telephone is found to be in use, an alternative contact method may be selected. Alternatively, an attempt at contact may be delayed, as the user is assumed to be busy.

4.5.1 Deployment

Two telephones have been equipped with a reed switch and magnet. These are both present in room Au501 in the CL. One telephone is an ORL extension, the other a CL extension.

4.6 Motion Detection

Medusa[WGH94] is a peer to peer architecture for controlling networked multimedia devices. The Medusa architecture consists of an ATM network, a set of multimedia peripheral modules, such as cameras and loudspeakers which are connected directly to the network, and a set of software modules which enable video and audio streams to be displayed, buffered and analysed.

To build applications, Medusa modules are connected together. For example, the elements of one-half of a video-phone application would include a source video feed from a camera module, a buffering module to smooth network jitter, and a sink module that would receive and display the networked video. Applications in daily use at ORL include a video-phone, a video-mail system and a terrestrial television viewer.

Figure 4.5 shows a typical arrangement for a Medusa equipped workstation. The workstation screen itself provides the video output. A camera is mounted above the workstation and another to the side of the workstation. Loudspeakers and microphones are also provided. All are connected via hardware interfaces to the ATM network. A database maintains relationships between peripherals so applications can, for example, determine which cameras and speakers to use when starting a video-phone session on a particular workstation.

Medusa modules can also provide non-interactive functionality such as face recognition, hand tracking and motion detection. It is this latter functionality that is made use of here.



Figure 4.5: Typical Medusa Setup

4.6.1 Detecting Motion from Video

Figure 4.6 shows schematically how video motion information is propagated to applications. The server is responsible for constructing a motion detector for



Figure 4.6: Motion Detection System

each camera by constructing appropriate Medusa modules and streams and fitting these together. The motion detection module uses block-matching motion estimation to produce motion vectors. If a motion vector is produced which is above a predefined threshold the server is informed. This motion information may then be communicated to applications.

4.6.2 Deployment

Motion information is available from 20 ORL cameras.

4.7 Summary

This chapter has described a diverse set of sensor devices suitable for monitoring aspects of a dynamic office-based environment. All have been deployed to some degree. At present, applications must use ad-hoc techniques to gain access to the information.

The representation and processing of this information is one of the the central themes of this thesis. Attention is now turned to architectural issues which enable each sensor system to be given a generic interface and also attempt to ease the task of building event-driven services and applications.

Chapter 5

Event Service Architecture

This chapter presents a general paradigm of distributed data delivery, which is suitable for the abstraction of the sensor systems presented in chapter 4. It is described how this paradigm is used, together with a standard distributed systems architecture, to build CALAIS *event services*, which provide uniform interfaces to heterogeneous sensor systems. Together with persistent resources these form the basic components of CALAIS.

5.1 Events and Event Service Requirements

Two main options exist to distribute information from sensor systems to interested clients:

• Polling

The client periodically polls the sensor system by submitting an enquiry regarding the state of one or more of the managed sensor devices.

• Asynchronous events

The sensor system informs applications whenever a new sensor reading is available. This information may be distributed by *multicast*, or by *callback*. In the former case, every item of sensor information is distributed to all applications which then discard unsuitable items. In the latter case, the application explicitly *registers* with the sensor system providing a reference to a call-back handle. The sensor system then invokes each call-back handle, parameterised with the sensor reading, every time an item of sensor system information is available.

Polling is not suitable for the delivery of sensor system information. It is wasteful of system resources both in terms of computation and network bandwidth. Furthermore an application must decide how regularly to poll, which assumes the application has prior knowledge of the sensor system characteristics.

Asynchronous delivery is the preferred option. Multicast is simple, but is also wasteful of network bandwidth and requires the application to determine the type of each event as it is delivered and discard those it is not interested in. The most suitable method of information delivery is that provided by the registration and call-back event paradigm. Information is sent to applications only when new information is available. No prior knowledge of sensor system behaviour is required by the application and the application only receives those types of event it is interested in.

5.1.1 Event Definition

Generally:

An event of class E has a set of attributes $a_1, a_2, ..., a_n$ and a timestamp t

Each event is of a particular *class*. Each event class may have zero or more attributes, such as the user-id associated with an Active Badge sighting. Each event is a discrete occurrence, that is it occurs at a specific point in time. Every event therefore has an associated time-stamp. This allows temporal comparison of events, based upon the time at which the event occurred rather than the time at which the application received notification of an event. This is an important requirement as the delivery of events within a distributed system may be subject to delay.

5.1.2 Event Service Requirements

As illustrated in chapter 4, sensor systems are implemented using a variety of technologies, and each has an ad-hoc interface. This places an unnecessary burden on the application developer. Event services should have a standard interface which can be added to any existing sensor system, providing generic registration and call-back facilities. This interface should be straightforward to define and integrate, and enable distributed access. These requirements result in event services having the structure shown in figure 5.1.

5.2 A Standard Distributed Architecture

CORBA (Common Object Request Broker Architecture)[OMG95], defined by the OMG (Object Management Group) is the distributed systems architecture used for CALAIS. The motivations for choosing CORBA are:

- OMG *interface definition language* (IDL) provides a standard interface to systems implemented using different operating systems and implementation languages.
- Location transparency of distributed components.
- Wide availability within the target environment.

40



Figure 5.1: Generic Structure of an Event Service

CORBA is a object-oriented, peer-to-peer architecture; the distinction between clients and servers made in other distributed system architectures, such as DCE^1 is not made. However for the duration of a method invocation, the calling object can be termed the *client* and the object receiving and performing the method invocation the *server*.

CORBA consists of three architecture areas: the *object request broker*, *object services* and *common facilities*. Parts of the first two are pertinent to this thesis and therefore are now briefly described.

The Object Request Broker

The object request broker (ORB) functions as a communications infrastructure, transparently relaying object requests across distributed heterogeneous computing environments. Both clients and servers communicate with the ORB when setting up a method invocation. Client and server objects may exist within the same address space, on the same machine or on different machines. The ORB allows applications transparent access to objects, irrespective of their relative physical location.

Two methods exist for CORBA objects to communicate: static interfaces, defined using an IDL, and the *dynamic invocation interface* (DII).

OMG IDL is a programming language independent syntax for describing application interfaces. IDL is compiled into programming language dependent stubs which enable the application to talk to the ORB. Both the client and server need access to these stubs and thus the application developer must have access to the corresponding IDL.

¹The Open Software Foundation's Distributed Computing Environment.

By using the DII, an application does not require *a priori* knowledge of an object's interface. Static IDL is not used. Rather an application accesses a system database of known interface signatures and then performs a generic method invocation to a server which has implemented the required object type. Such a facility is useful to applications which may not know in advance the interface of an object they wish to communicate with.

By the use of an interface language, CORBA hides the implementation of an object. It is therefore suitable for abstracting away the details of a sensor system. An application need not know how an sensor system is implemented, what software and hardware is involved and which operating system it is supported by.

Object Services - Naming Service

Object services comprise a set of system service interfaces which provide commonly needed facilities for application and system developers. These services are collectively known as the *Common Object Service Specification* (COSS)[OMG96]. The naming service is of particular interest here.

To initiate contact with a CORBA object, which perhaps provides access to a CALAIS event service, an application must obtain an address for that object. Such an address will typically encode the name of the machine where the object is located and other addressing information. This address may change often, for example, when the process providing the object is restarted. The solution is to associate a name with such an address and enter this information into the COSS naming database which applications may then query. A name is multi-level, for example CALAIS/sensors/active-badge. Each time the service providing such a named object is started, the corresponding object reference is exported to the naming service. An application can then fetch the object reference and then contact the object directly.

5.2.1 Inter-ORB Communication

As part of the CORBA 2.0 specification the OMG have defined the *Internet Inter-ORB Protocol* (IIOP). This enables ORBs from different vendors and residing within different Internet domains to communicate. An application supported by one ORB can communicate with an object supported by another ORB in the same way as it would communicate with a local object. This further extends object location transparency.

5.2.2 Local System Setup

At ORL where most of the implementation work within this thesis has been carried out, CORBA is present on multiple Unix platforms, Windows NT and ORL's locally developed operating system, ATMOS. The CORBA 2 implementation omniORB is one developed locally at ORL. omniORB uses IIOP as its

native transmission protocol. A COSS naming service is also available.

Other implementation work has been carried out at the CL, again using CORBA. Here the ORB implementation used has been IONA Technology's *Orbix*. By using IIOP technology, objects at ORL and the CL can communicate.

5.3 The Structure of CALAIS Event Services

The structure of an event service is illustrated in figure 5.2.



Figure 5.2: Structure of an Event Service

Each CALAIS sensor system is extended with a CORBA interface, specified in IDL. This IDL provides a description of the attributes of the event class and specifies the client interfaces to each event source. The resulting *event services* provide a standard set of interfaces and facilities.

5.3.1 Registration and Call-back

As previously stated, a registration and call-back paradigm is adopted for the delivery of events. The event service provides interfaces to allow an application to *register* and *de-register*, and an interface to deliver events to applications using call-backs.

Event Templates

A client may not be interested in every event that a service offers. Consider the *xmonitor* event source, previously described in chapter 4, which produces activity events from workstations. A client may only be interested in events from a particular workstation. It is convenient if event occurrences can be *filtered*. Event filtering could be performed by a client using a provided library of suitable routines. However this approach will result in all events being delivered to every client. To reduce this wasted network bandwidth and the resulting increase in client complexity, each event service performs filtering before the event is delivered. To perform this filtering the service holds an *event template* for each client. This template is provided by the client at registration time. If, for example, the client is interested in *xmonitor* events but only those produced by workstation 'scallop' the following event template may be given by the client:

$workstation \ name$	workstation domain	user- id
scallop	cam-orl.co.uk	*

The * indicates a wild-card and will match any attribute value. Before an event service notifies an event occurrence to a client it compares the event to the appropriate template. Only if it matches is the client notified.

By default, event attributes are tested against a template using an equivalence operator to determine whether an attribute is exactly the same as the corresponding template attribute. The equivalence operator used will be determined by the type of attribute. Using equivalence is not suitable in all circumstances however, and therefore other types of binary operator can be chosen when constructing an event service.

5.3.2 Time-stamps

Each event has an associated Unix style time-stamp consisting of the number of seconds and micro-seconds since 1st January 1970.

5.3.3 Storing and Querying Events

Events are discrete temporal occurrences. An event can be considered a report in the change of an object's *state*. Some event services will produce events at regular intervals, others irregularly. Furthermore the mean frequency of event reporting will differ from service to service.

In many circumstances it is not appropriate for an application to wait for the next event from a service, and it may require to know directly the last event that occurred which matches a particular event template. Consider for example an application which displays a person's unread e-mail on a workstation when they walk into a room. A condition for the e-mail to be displayed is that the workstation is not currently active. Activity events, made available by the *xmonitor* event service, are only produced when a workstation is active and therefore the application must query the event service to find out when the last activity event occurred for that particular workstation. Only if the event occurred more than a time interval T ago will the e-mail be displayed.

The approach taken here is to make an attribute of a service's event class *primary*. Then, for each differing value of this primary attribute the most

recent corresponding event is stored. For example, for the *xmonitor* service, the primary attribute is workstation name. Thus for each workstation the most recent corresponding activity event is stored.

A synchronous interface is provided to the event service which takes an event template as an argument. Using this, the stored events are queried and those which match the template are returned immediately.

It can be argued that a greater depth of state should be available. However, if required, an event history can easily be created by a client which registers with the appropriate event service and then simply records each event as it is received. Current work within the CL is investigating a novel database architecture for the storage of events[BS98]. This allows the replay and off-line analysis of event histories.

Service Persistence

Depending upon the frequency of the event class a service supports, it may be appropriate to make an event service *persistent*, that is store each most recent event on a permanent medium such as disk. If the service then fails recovery is possible. Using a persistent store is only suitable for event services where the frequency of events is low, for example in the case where an event may only occur once every hour. It is less suitable for services where the event rate is high, for example where several events occur per second, as not only would the persistently stored events be overwritten quickly after recovery, but the frequent writing of event records to disk will cause an extra and unnecessary system load.

5.3.4 Dynamically Discovering an Event Interface

Each event service also offers a generic interface to the events it offers. This *type-less* interface allows an application to be written which does not know a *priori* the class of events which it is interested in receiving. Also, an application which does not have access to an event service's IDL can use this interface. By accessing an *event class repository*, a service which maintains a database of event classes known to the system, an application can receive the name of the event, a description of the event's class, and an object reference enabling it to contact the type-less interface of the event service.

This process can be compared to CORBA's DII (which is not currently implemented in omniORB) which similarly allows dynamic binding to object interfaces.

Figure 5.3 shows the process by which an application dynamically discovers the class of an event. The application contacts the event repository and issues a request for information about a particular event class name. It then receives a description of the event class, that is how many attributes the event has and the types of these attributes. It also receives an object reference which the application can then use to contact the event service directly.



Figure 5.3: Dynamically Discovering an Event Class

The event class repository is in fact an event service like any other, although lacking the type-less interface. The event repository therefore offers call-back and query interfaces. An application such as a browser displaying information about an organisation's event services, may register interest in additions and updates to the repository's database. The repository is an example of an event service which requires persistence.

5.4 Building Event Services

CALAIS event services all share the same common features. Each has a registration and de-registration interface and passes data to clients using call-backs. Each stores a set of recent events available for querying. All that differs between services is the event class that it offers to clients. By recognising these common features it is possible to produce service implementations automatically by providing a generic server library.

Figure 5.4 shows schematically the process of building an event service. The process starts by the service developer declaring an event class in IDL. This is then augmented by declarations for the standard interfaces. Using this IDL, CORBA stubs are produced and program code generated to handle data management tasks. The service developer must supply a template comparison function and an interface to the low level system which actually produces the events.

By the use of appropriate tools this process can be automated. This leads to significant reductions in the time taken to develop an event service.

Constructed Event Services

This technique of constructing event services is applied to each event source described in chapter 4. IDL for each event service can be found in appendix A. Table 5.1 shows each event class, with its name and associated attributes.





5.5 Database Support Services

CALAIS system components and applications require support from database system resources. These differ from event based sources of information in that the data held typically changes infrequently. Also, such resources may come from databases that already exist and currently provide support for other systems.

Some examples of required information resources are:

- 1. Equipment capabilities: information about equipment such as its type, network interface, screen size, available input devices etc.
- 2. User preferences: examples include the circumstances under which a user is able to receive e-mail and take phone calls, the name of their office, and an event expression which when satisfied leads to an application being invoked.
- 3. Layout of the physical environment: location aware services need access to information about the physical environment. This may include where doors and offices are situated and the location of stairs.

Such databases may deal with less volatile information, but may still be regarded as sources of events. A particular type of change to a database entry may be defined as an event class and interested clients may then receive such updates by registering in the usual fashion. Many traditional DBMSs have a form of event call-back termed a *trigger*[WC96]. Such triggers may also be defined as event classes which inform an application of the change itself, or simply that a change has taken place. The latter method may be used where a database is of sufficient complexity to require interrogation using a query language such

Event Source	Event Class	Event Attributes
Active Badges	ABadge	badge type: either 'person' or 'equipment' user-id: identifies the owner of the badge user domain: identifies the user's domain room-id: identifies the room which the badge is within room domain: the domain of the room button click: either 'side', 'middle', or 'no-click'
Ultrasound Badges	UBadge	<i>badge identifier:</i> uniquely identifies the badge <i>x</i> , <i>y</i> , <i>z</i> : a 3D coordinate relative to a coordinate system <i>coordinate system:</i> the coordinate system identifier
Workstation Activity Monitoring	WSactivity	<i>workstation-id:</i> the host name of the workstation where activity occurred <i>workstation domain:</i> the domain of the workstation <i>user-id:</i> the user identifier of the person using the workstation
Door Position	Door	<i>door identifier:</i> the name of the door. <i>door domain:</i> the domain of the door <i>status:</i> either 'opened' or 'closed'
Telephone Handset Status	Phone	<i>telephone-id:</i> the extension number of the telephone <i>telephone domain:</i> the name of the telephone's domain <i>status:</i> either 'pick-up' or 'put-down'
Motion Detection	Motion	<i>camera identifier:</i> uniquely identifies the camera <i>domain:</i> the camera's domain

Table 5.1 :	Event	Class	Attributes
---------------	-------	-------	------------

as SQL. It is inappropriate and inflexible to attempt to define every type of database update or query as an event class.

5.6 Component Summary

Figure 5.5 shows the components of CALAIS presented so far. Details of the location service and composite event detection are given in later chapters. A description of database services used will also be given where appropriate.

5.7 Distributed Time

Two issues regarding time within a distributed system are of interest:

• No global time

Each machine within a distributed system has its own clock. Each clock may drift at varying rates.



Figure 5.5: CALAIS Components

• Network delay

Messages sent between machines will incur a delay dependent on machine and network loads.

5.7.1 Clock Synchronisation

Consider an application receiving events from two different services. These services will typically be running on different machines, each of which has its own clock. If no clock synchronisation is performed the application will be unable to determine an ordering of events. A number of approaches have been developed to address this situation.

The Global Positioning System (GPS)[AL94] transmits Universal Time Coordinated (UTC) signals which have an accuracy of between 0.1-10ms. Commonly, clocks are synchronised within the local area using the Network Time Protocol (NTP)[Mil91]. This achieves an accuracy of below 100ms even in wide area networks.

A pragmatic approach is taken towards clock synchronisation here. At both sites used in development (ORL and the CL) NTP is used. Indeed an NTP server at one domain peers with another server at the other domain. This ensures that time-stamps generated within the two domains will be synchronised to within 10ms.

Schwiderski [Sch96] points out that some distributed systems require high accuracy time-stamps to be applied to events. An example cited is a distributed debugging system producing up to 500 events per second. In the application domain considered here such high frequencies will not occur and therefore a time-stamp accuracy of 10ms is considered adequate.

5.7.2 Network and Registration Delay

The transmission of an event over a network may cause a delay in its delivery. Consider an application receiving two events a and b, from two event services A and B. The event a, dispatched to an application before event b, may in fact arrive at the application after b due to network delays. The application may therefore not respond as expected.

Another problem is that registration with an event service is not instantaneous. Therefore it is possible that between the time at which an application initiates registration and the time at which registration is complete, an event occurs which is not issued to the application.

These problems are both non-trivial to solve. Both will be considered further, and appropriate solutions proposed, in chapter 9, where client support for dealing with multiple event sources is discussed.

5.8 Naming and Inter-domain Issues

It has been shown how event services can be built to provide uniform interfaces to applications. Howevever, the use of multiple event services may still be hindered due to one event service naming a semantically equivalent event attribute in a different way to another service. CALAIS uses system independent naming. Therefore each event service is responsible for converting internal, system dependent names, to their system independent equivalents.

People, for example, are represented by their system login name. This is unique within a domain. Consider, for example, the *Active Badge* and *xmonitor* event services. xmonitor directly produces events with a system login user-id. The low-level Active Badge system directly produces a badge-id. In the case where a badge is monitoring the location of a person, both these identifiers refer to a person. In the case of the xmonitor event service, people are already represented with a user-id. For the Active Badge event service, conversion is performed between badge-ids and user-ids by adding a middle layer between the event source interface, described in section 4.1, and the CALAIS client interface. This maintains a lookup table of badge-id, user-id pairs. A similar approach can be applied to other types of object.

5.8.1 Inter-Domain Working

It is beyond the scope of this thesis to consider in detail the naming and information exchange issues that must be solved to enable the mobility of physical objects between multiple administrative domains. Although consideration of these issues has been made in the design of CALAIS, the separate domains of ORL and the CL are generally treated as parts of the same domain.

To enable inter-domain working, a method must exist for the distribution of object names and object attributes to different domains, thereby allowing an object to identify itself when it enters a domain. This relies on the object itself internally containing naming information which is globally unique.

Three possible approaches for the distribution of object names and attributes are:

• Explicit representation within each domain.

The identities of mobile objects are added in each domain the object is likely to encounter. This information exchange is likely to be managed by a system administrator.

• Federated naming scheme.

Each domain exists in a hierarchy, in a similar way to DNS[Moc87]. An update in one domain causes the update to be progressively propogated throughout the hierarchy, eventually reaching every other domain.

• Agent scheme.

The unique address provided by the mobile object entering a domain will trigger an agent. The agent will then attempt to contact a *home domain*[IB94] to obtain the attributes of the object.

5.9 Summary

This chapter has described a CORBA based framework to provide CALAIS components with a uniform communications infrastructure. A methodology has been described for the building of event services from generic components. Each event service offers standard interfaces to applications. These interfaces allow registration and state querying. Both typed and type-less interfaces are provided. An event trading service is presented which enables event classes and services to be discovered dynamically. The requirement for access to database information was identified and an indication given when the event model is suitable for propagating database updates to applications.

52
Chapter 6

Location Service Requirements

This chapter identifies the requirements that a location service, suitable for the management of location information, must fulfill. The design and implementation of the location service is then described in chapter 7.

6.1 The Significance of Location

Location is a well known everyday notion. The location of physical objects may be described in a variety of ways:

- "The desk is in the study".
- "Giles is near the window."
- "The workstation is at coordinate x, y, z".
- "The ship is at longitude R and latitude S."
- "The screen is oriented at 12 degrees."

Location information within the context-aware domain has been considered earlier in this thesis in chapter 4, which described the Active Badge and ultrasound location systems. Used directly, this information is of only partial use. With knowledge of the location of the object, and some sense of the structure of the location domain, a user can physically find the object and then interact with it.

However, many more applications become possible when knowledge of the location of many objects is available. Consider the following example. A software agent is responsible for the timely delivery of video-mail to a user. To carry out its function the agent tracks the location of the user and determines the items of equipment that are nearby, perhaps within a distance of 2 metres. The equipment list is narrowed by the application determining which items of equipment are capable of delivering both video and audio. Once a suitable item of equipment has been found, the agent can then attempt to deliver the video-mail.

What is significant about this example is that the application is not interested in knowledge of the *absolute* position of the user or equipment. Rather, it is interested in the names of those objects which are nearby, or *co-located*[Nel96] with the user.

The responsibility for evaluating co-location queries could rest with applications which would gather and analyse location data in ad-hoc ways. However it is asserted that all applications can benefit from a set of common features and these should be provided by a *location service*.

6.2 Requirements

6.2.1 Location Information Integration

A location service should be capable of integrating multiple types of location information. To have the widest applicability, three spatial dimensions must be supported. This leads to a requirement for a generic view of location information. A location can be defined as follows:

A *location* is the name of a *coordinate system*, a *position* within this coordinate system and an *error* in this position.

Within a three-dimensional Cartesian coordinate system the position and error will be represented by two triples. The first (x, y, z) represents the object's position. The second (x_e, y_e, z_e) represents the maximum error in each dimension relative to this position.

Using this definition the following sources, amongst others, of location information can be represented:

- Active Badges;
- Ultrasonic devices;
- Manual measurement.

Ultrasonic devices and manual measurement give a position directly. Active Badges however, directly give a position within a container, an infra-red zone (see section 4.1) which is typically regular in shape. This information is termed a *containment*:

A *containment*, given by the name of a container, indicates the presence of an object at some unspecified position within the container. A containment can be represented as a location by specifying the position to be at the centre of the container, and the error equal to the distance from this position to the edges of the container. Thus the location specifies the position of the object at the centre of the container, but indicates by the error that the true position may be anywhere within the container. This is illustrated in figure 6.1.



Figure 6.1: Position and Error Representation in a Container

Each location sensing system will typically operate within its own coordinate system. The ultrasonic badge gives a position relative to the room in which it is installed. Manual measurement of the position of objects will typically be done relative to a room or part of a room. To enable the positions of objects within different coordinate systems to be compared, each item of location information requires transformation from a sensing coordinate system to the location service's own coordinate system.

6.2.2 Physical Object Representation

All physical objects have a non-zero, finite, spatial extent. Together with its location, an object's extent determines how it will interact with other objects. These extents should therefore be represented within the location service. In a computer graphics system, where objects are required to be realistically rendered, the spatial entity used to represent an object would typically be a complex one. However in a location service where no rendering is required, where location sensors are limited in accuracy and where application requirements are less exacting, it is possible to use a simpler representation.

Non-Physical Objects

Objects stored by a location service do not necessarily have to exist physically. Examples of such objects are:

- A radio field.
- The audible range of a telephone.

The location of such objects will typically be derived from the location of a physical object. In the case of a radio field this might be the location of a radio transmitter.

Knowing those objects co-located to such non-physical objects may be important for some applications. For example, an application may wish to know when a particular person is in audible range of a telephone, in order to redirect a telephone call.

6.2.3 Types of Query

It is asserted that context-aware systems are interested in three types of locationoriented query:

- 1. where is object A?
- 2. what objects are within spatial region R?
- 3. what objects are within spatial region R, where R is defined relative to the position of object A?

For some applications the ability to synchronously execute the above types of query is not sufficient. In a similar way to earlier discussions of polling versus call-back in the context of event services, the polling of a location service by regular query submission is inefficient. Applications may need to respond quickly to a change of context and polling may result in a delay as well as an unnecessary increase in system load.

Therefore, for each of the query types described above an asynchronous equivalent is also required. These can be expressed as follows:

- 1. keep me informed where object A is.
- 2. keep me informed of the objects within spatial region R.
- 3. keep me informed of the objects within spatial region R, where R is defined relative to the position of object A.

6.2.4 Orientation

In addition to a position, physical objects also have an orientation. By representing this the results of location based queries can be enhanced with information about the relative orientations of objects. In some circumstances, interaction with physical objects is only appropriate when the target object and the interacting object are oriented with respect to each other in a particular way. For example a person may only interact with a workstation when the person is in front of, and facing, the screen.

Generally orientation can be represented with a normal vector defined in relation to a particular object face[Fra91]. Operations such as the dot product can then be applied, for example, to determine the angle between two normal vectors and thus find their relative orientation. Orientation complicates the method by which location information is gathered. To determine an orientation in two-dimensions, the position of two distinct non-collinear points on the object must be measured. For a three-dimensional orientation three such measurements are required. To measure the orientation of mobile objects therefore requires multiple location devices per object and greater software complexity to interpret the multiple location sightings. A fine granularity of location information is required to determine orientation accurately.

6.2.5 Performance and Scale

This section attempts to identify a set of performance requirements a location service must fulfill. These can then be borne in mind during design and implementation. The quantitative aspects of this are inevitably imprecise, and should be interpreted only as orders of magnitude.

Real-time Response

A context-aware system is a live, event-driven system operating in real-time. Many applications must respond to an event quickly to be effective. A typical application operating within a mobile environment will require a response time of not more than 1 second.

Although the issue of guaranteeing real-time response to applications is beyond the scope of this thesis, steps should be taken to minimise query latency.

Scale

The number of physical objects which the location service must represent will have a direct impact upon performance. It is therefore important to attempt to estimate this number. This will be done for the domain of ORL.

ORL contains a staff of approximately 50. Each person is currently equipped with an Active Badge. Equipment badges are used on approximately 130 items of equipment. Imagine if the number of objects to be monitored increases tenfold. This may include more items of equipment such as telephones and even such things as books. This estimate leads to a requirement for the location of approximately 2000 objects to be supported. Typically, sets of objects will be updated at different rates, depending upon their rate of mobility. A figure of 500 updates per second may be realistic.

Further to this, the number of applications requiring query facilities must be considered. As the number of objects tracked increases and the location information resource becomes larger, it is logical that the number of queries will increase. Again, to produce an order of magnitude, it can be assumed that one query per tracked object will be active at any one time. This therefore produces a requirement for 1000s of queries to be handled. Many of these will require asynchronous call-backs, triggered when an update is performed.

Summary

The performance estimates given here should be borne in mind in the development of a location service. The handling of thousands of operations per second implies that a model and implementation need to be highly efficient.

6.2.6 CALAIS Integration

The location service must integrate with other components of the CALAIS architecture. It will be a client of location sensing services and act as a service to applications interested in issuing location queries. As with other services, CORBA interfaces will be available. In chapter 8, it will be shown how the results of location queries can be integrated into the event paradigm and so enable convenient access for applications.

6.3 A Spatially Equipped DBMS

To explore the implementation requirements of a location service it is worth considering if a commercial data processing system, equipped with spatial capabilities, could be used to realise a location service.

Illustra[Ill95] is a database management system which can handle spatial data directly. Illustra¹ is the commercial version of Postgres[SRH90]. It is termed an *object relational* system and uses an extended version of SQL (Structured Query Language)[Can93] to provide a limited amount of object oriented functionality in addition to standard relational facilities.

Illustra has a meta layer which stores information concerning known data types, functions and indexing methods. This meta layer makes Illustra extensible and new types, functions and indexing methods, provided within libraries, can be added, thereby extending the system's functionality. Illustra terms these libraries *datablades*. Datablades are available which provide specialist functionality for the management of two- and three-dimensional spatial data. These provide a set of geometric types, such as rectangles, circles and polygons, and functions, such as intersection and containment. Furthermore, spatial indexing is provided by use of an R-tree data structure.

Other relevant facilities which Illustra provides are:

• Server execution of user-supplied functions. User-defined functions which use database facilities normally reside in a separate client process which communicates with the database server via a network. If required, however, a function can be dynamically loaded into the server process and then invoked remotely. This is suitable for functions which are used often

¹At the time of writing Illustra Information Technologies is now part of Informix Software. Illustra itself is no longer available but its technology has been used in Informix's Universal Server.

and where performance is a priority. It is this facility that enables Illustra to be extended with datablades.

• Support for asynchronous notification of events. Illustra supports the ECA (event-condition-action)[WC96] model. Rules can be defined on database tables and an event defined which will be invoked when an action occurs. This can notify a remote client which is waiting for event occurrences.

Illustra also supports standard DBMS facilities such as transactions and recovery mechanisms.

The combination of the above facilities make Illustra appear very suitable to act as the core of a location service implementation.

Illustra Evaluation

Illustra must show that it can cope with a high update rate. To evaluate Illustra's performance characteristics, a database was defined suitable for the storage of Active Badge information, including a table suitable for holding raw sighting information consisting of approximately 10 attributes. In use this would be updated many times per second. An initial sighting was inserted into the table for approximately 50 badges and an Active Badge feed created which updates the table with the most recent sighting from each badge. The results, shown in table 6.1 were produced on a Sun SPARC machine with a 50Mhz processor, 10ms seek time disk, and 128Mb of main memory.

	no index	with index		
update	101ms	109ms		
search	28ms	28ms		

Table 6.1: Illustra Update and Search Performance

These results show the time required for an update or search, with and without a single B-tree index having been defined on the primary attribute *badge id*. Each update causes a disk write. No datablade extensions are present.

Conclusion

Unfortunately these results indicate that Illustra is not suitable to be considered as the storage and indexing technology for a location service. The 101ms update time with only 50 tuples present in the table means only 10 updates can be performed per second, ignoring any other work the database system may have to perform. It is unclear why Illustra's performance in this simple example is so poor. The search results indicate an upper limit of 28ms on the time taken for query compilation and the determination of the correct tuple to update. The remaining 73ms required to perform an update indicates the transaction processing system may be causing the performance bottleneck.

It appears a general purpose DBMS lacks the performance required. An alternative, lightweight, approach is described in the following chapter.

Chapter 7

Location Service Design

The use of a general purpose DBMS, equipped with spatial processing facilities, to realise the location service, was rejected due to the poor performance encountered. In a response to this a lightweight method of location information management is now proposed, aimed at providing high performance.

A method of representing the location of physical objects using a simple geometric shape is firstly described. Then, suitable indexing technologies are reviewed and the most appropriate method chosen. An approach to representing the relative orientations of objects is presented, followed by methods of representing a domain's static environment. It is then described how powerful spatial queries can be evaluated and how this functionality is made available to applications.

Figure 7.1 summarises the proposed architecture of the location service.



Figure 7.1: Proposed Location Service Architecture

7.1 Physical Object Representation

In the following discussion, without loss of generality, two-dimensional terminology is used.

The aim of the location service is to represent the locations and spatial extents of physical objects, thereby allowing the execution of location based queries. This internal representation will minimally consist of the *name* of the object and a *spatial entity* representing its position and extent.

An internal representation of *minimum bounding rectangles* (MBRs), aligned to coordinate system axes, is proposed. An MBR is the smallest rectangle which can completely contain an object. As rectangles are simple, manipulation is efficient, and as will be seen, rectangles can also be used directly with indexing technology thus further enhancing performance. It is asserted that axisalignment is a minor restriction, and an aligned MBR is capable of providing a good approximation to the real extent of a physical object.

Figure 7.2 shows a schematic view of an office and the set of objects contained within it, the spatial extent of each represented using an MBR.



Figure 7.2: Object Representation within an Office

Two people, P_1 and P_2 , and various items of equipment, E_1 to E_4 , are depicted. The equipment might consist of desks or computers. Each one of these MBRs is aligned to the coordinate system axes. The MBR's extent and position is derived from a database holding the dimensions of physical objects, and its location from some location event source.

Now consider figure 7.3 which illustrates how a spatial search might be conducted. In this scenario an application wishes to know the objects that are within a distance of 2m of person P_1 . A circle is shown, centred at P_1 's centrepoint and of radius 2m. Depending upon the way in which the query is evaluated, the query may return those objects that intersect, or are wholly contained within, the search extent.

7.1. PHYSICAL OBJECT REPRESENTATION



Figure 7.3: Object Interaction with Searching

7.1.1 Handling Location Errors

Each item of location information is composed of a position and an associated error. When the position is matched with the object being located this forms a spatial extent formed by the position, the extent of the object and knowledge of where the location device is positioned on the object. Consider figure 7.4. The



Figure 7.4: Handling Errors

error of the location is represented by the dashed rectangle. This error extent is *added* to the spatial extent of the object itself, resulting in the MBR which is actually stored, shown by the dotted rectangle. A spatial query will therefore return a set of *candidate* objects which *may* satisfy the search predicate. As will be described, the results of a query can be filtered by the application specifying a maximum acceptable error.

7.1.2 Record Attributes

Each physical object has an associated object record stored within the location service. This has the following attributes:

- the name and type of the object;
- an MBR representing the object's spatial extent and position;
- the maximum error in the position of the object in each dimension;
- an MBR used in spatial searching formed from the addition of the MBR representing the object's extent and the maximum error in the object's

position;

• the time-stamp of the location report.

A single record will be held for each object. This will contain the most recent location information available for that object.

7.2 Indexing Technologies

The query types, outlined previously in chapter 6, require object records to be searched by name, and by spatial position. The number of objects being handled is large and therefore *indexing* is required to enable fast searching and updating. Spatial indexing is a well-researched area and is used in many application domains such as computer graphics and GIS.

Spatial data has multiple attributes, one for each dimension. These attributes, although representing orthogonal dimensions, do have dependencies between them expressed in Euclidean space by the Euclidean metric. Worboys[Wor95] illustrates by example that a traditional uni-dimensional indexing technique is not capable of indexing satisfactorily multi-dimensional dependent data. A type of index is required that can take advantage of ordering in two or more dimensions. Range searches in particular will benefit from an organisation which leads to those entities which are near one another in space, being near one another within the index. The following discussion is based on Worboys' classification of raster, point and complex spatial objects. Again, two-dimensional terminology is used but the principles equally relate to three-dimensional objects.

7.2.1 Raster

A raster is an array or grid of cells, commonly referred to as pixels. Rasters are capable of representing many spatial objects: a point may be represented by a single cell, and a connected area by a set of contiguous cells. A widely used structure for holding raster data is the *region quadtree*. It is based on the repeated subdivision of a raster into four equal-size blocks until each block is of a preset minimum size. An example of such a decomposition is shown in figure 7.5.



Figure 7.5: Space Decomposition Using a Quad-tree

The advantages of the region quadtree are that it takes advantage of the multidimensional nature of the data and it has variable resolution, only increasing tree depth in places where necessary detail is required. However a small translation in the raster may result in a large change to the quadtree structure - this results in inefficiencies for the representation of spatial objects in a contextaware system, the positions of which are dynamic in nature.

7.2.2 Point-Based

A spatial shape can be represented by one or more representative points. This may be the centroid of the shape, or in the case of a rectangle for example, the (x, y) coordinates of the two diagonally opposite corners. This second representation can be indexed using a k-d tree[Ben75]. The k-d tree is a k dimensional search tree where, at each level of the tree, a different coordinate is tested to determine which branch of the tree to take. In two dimensions, for example, x coordinates would be compared at the root (level 0) and even levels, and the y coordinate at odd levels. Two rectangles intersect when their projections on the x axis and y axis both intersect.

7.2.3 Complex Objects - Rectangles

Rectangles in particular have been the focus of a significant amount of research into indexing techniques. A thorough review of this work is presented in [Sam88].

The *R*-tree[Gut84] is a multi-dimensional extension of the B-tree, the latter commonly used to index uni-dimensional data. In an R-tree, leaf nodes contain the MBRs associated with the actual objects being indexed. Internal nodes hold the smallest MBR that contains all rectangles in its subtree. Like the B-tree the R-tree is self-maintaining and has $O(\log n)$ insert, delete and search complexity. A simple example of R-tree decomposition is shown in figure 7.6. A variant of the R-tree is the R+-tree[SRF87]. This tackles a problem with the



Figure 7.6: Space Decomposition Using an R-tree

conventional R-tree structure, in that non-leaf, container rectangles may overlap. Inefficiencies may therefore arise when searching for objects as the search may need to be performed in more than one subtree, even though the object will only be found in one of them. The R+-tree does not permit overlapping container rectangles. It achieves this by decomposing object MBRs if necessary. As Samet[Sam88] describes, this results in an increase in the height of the tree, but searching times are decreased. However insertion times are increased as an object may have to be decomposed and inserted into many subtrees.

7.3 Location Service Core

Figure 7.7 illustrates the core of the location service.



Figure 7.7: Basic Architecture of the Location Service

A single record is held for each object, which holds the last reported location of the object. To ensure high performance, records are held in main memory and are never written to a secondary store. The durability of data is not ensured therefore in the event of a system failure. This is acceptable because:

- A single record is held for each object. Location data is highly dynamic and therefore the lifetime of a record is short.
- The location service can request information stored at location event services at initialisation.
- It is estimated that a single location record, together with an associated index entry, requires approximately 200 bytes. With the continuing fall in the price of RAM it is realistic to expect hundreds of megabytes of RAM to be available, so enabling millions of objects to be stored.

7.3.1 Indexing

Searching may take place by spatial area or object name. Two types of index are used to support this. A standard R-tree is selected to support spatial searching for the following reasons:

- Direct mapping between location service object representation and index primitive (MBR);
- direct support for range queries;
- $O(\log n)$ complexity;
- its low cost insertion and deletion operations compared to, for example, the quad-tree and also the R+-tree;
- self-maintaining.

Indexing by name enables the location of a object to be rapidly found. This is straightforward as name data is uni-dimensional. Furthermore a sophisticated self-maintaining structure such as a B-tree is not essential, as object names are relatively static, compared to the highly dynamic spatial data. A *hash* index is selected and an appropriate hashing function determined which provides good data distribution with a wide variety of names.

Both the R-tree and hash indices are held in main memory.

7.3.2 Coordinate Systems

The location service uses a single, three-dimensional Cartesian coordinate system. This enables all objects to be available in a search without the need for any internal coordinate transformations. The origin of the coordinate system will be at some reference point in the physical spatial domain, for example, the bottom corner of a building. It is not important where this actually is, provided an application needing to interpret object coordinates has access to this origin position.

Physical objects are represented using the three-dimensional equivalent of a rectangle, a *cuboid*, defined using two (x, y, z) coordinates. These represent the cuboid's opposite corners.

7.3.3 Interfacing to Location Sources

Each source of location information must be interfaced to the location service. This interface translates from the representation of location used within the location source to the canonical form used in the location service (section 6.2.1). Coordinate system transformation is also required. For example the Active Badge location service gives locations in terms of containment within an infrared zone. The interface therefore requires knowledge of the extents of infra-red

zones expressed within the single coordinate system used by the location service. Also each interface must have knowledge of the dimensions of the objects being located and knowledge of the positions of sensor devices on each of these objects. Figure 7.8 shows the components of a typical interface.



Figure 7.8: Location Service Interface

The databases holding coordinate system and object dimension information may be provided by any convenient mechanism, such as a general purpose DBMS. The maintenance of these will typically be a task carried out by a system administrator on a periodic basis. The mechanism used for interfaces for the Active Badge and ultrasound location sources is a disk-resident flat file with a CORBA interface.

7.3.4 Performing an Update

When an object is seen by the location service for the first time an *insert* occurs. An object record is created and an entry made in the hash and R-tree indices. Thereafter, each time the object is seen an *update* is performed, replacing the current object record with the new data. This update is performed as follows:

- 1. Find the object's current record using the name as a key into the hash index.
- 2. Use the current indexed MBR to perform a deletion from the R-tree.
- 3. Update the object record.
- 4. Determine the new indexed MBR by adding the error to the object's spatial extent.
- 5. Insert the new MBR into the R-tree.

This describes the basic updating process. Extensions which handle asynchronous query evaluation will be described below.

7.4 Representing Orientation

Section 6.2.4 outlined the way in which the orientation of objects may be represented and used. The approach to orientation taken here is one that fits closely with the axis-aligned MBR model of object representation presented so far.

Physical objects can be considered to have one or more *active* sides. One of these sides is typically oriented towards another object when interaction is taking place. Examples of active sides are the front of a person or the side of a monitor containing the screen; a chair with a back may have four active sides (considering three-dimensions), the back and bottom excluded.

Consider figure 7.9. This illustrates a set of objects within a room, represented



Figure 7.9: Active Sides of Objects

as before with MBRs. The active sides of each object are shown by bold lines. Each object has a single active side, apart from b which has all sides denoted as active.

An active side can be represented by two points. This encodes in two-dimensions a line and in three-dimensions a plane. By itself this enables determination of the axis along which an active side is oriented, but does not indicate the *direction* along this axis, positive or negative. To enable determination of this, the opposite side to the active side is also represented.

Now consider figure 7.10. The shaded area contains those objects which may



Figure 7.10: Determining Facing Objects

be considered to *face* the target object X. A *facing* object must firstly have an active side aligned on the same axis as an active side of the target object within the shaded area. Objects b, c, d and e qualify. The direction of each object must also be then examined. The direction of the active sides of objects b and e, within the shaded areas, are the same as X and are therefore *not* facing. The direction of objects c and d are opposite to that of X and therefore do qualify as facing objects.

The spatial area in which to examine for facing objects can be determined in a number of ways. The strategy used in figure 7.10 is to examine an area bordered on one axis by the active side and extending to the limits of other axes. An alternative and stricter policy is illustrated along with the original method in 7.11.



Figure 7.11: Alternatives for Facing Area

This stricter method ensures only those objects which are *directly* facing the target object are found. The decision on which method to use can be left to the application which can specify a choice when initiating execution of a query.

7.4.1 Expressing Orientation

The orientation of an object is expressed relative to the orientation of a target object. Primarily, applications are interested in whether objects are facing the target object. An application will receive an enumerated type for each object describing how the object is oriented with respect to the target object. The value of this enumerated type may be one of the following:

- 1. **facing:** any active side of the object is oriented on the same axis, and directed in the opposite direction, relative to any active side of the target object.
- 2. not-facing: condition 1 does not hold and the object has an active side.
- 3. unknown: the object has no active sides.

Here, only those objects *facing* the target object are determined. By extending the paradigm it would be possible for other types of orientation to be expressed, for example *above*, *below* and *behind*.

7.5 Query Evaluation

The location service provides rich spatial querying capabilities, in particular the asynchronous monitoring of spatial regions.

7.5.1 Spatial Queries

To execute a spatial query an application provides a rectangle and a flag indicating whether intersection or containment should be used as the selection criteria. The rectangle is used to index the R-tree. The list of object records with an indexed MBR which intersect, or are contained within, the search rectangle are returned to the application as a result. These are ordered with respect to distance from the centre of the search rectangle.

The application may also submit a filter which the location service applies to the query result. This is used in the same manner as the event templates described in chapter 5. In this context it may be used, for example, to filter records which have a location error above a certain threshold.

Proximity Query

A spatial query can also be submitted, relative to a target object. Instead of the application submitting a rectangle defining the space within which the search should take place, it submits a maximum distance in each dimension.

The location of the target object is firstly found and its centre point calculated. The search extent is then determined using this centre point and the application supplied distance. The error in the location of the specified object is also taken into account by the addition of this to the search extent. The search is then executed by searching the R-tree with the resulting rectangle.

After the set of objects intersecting the spatial extent has been found, the orientation of each one of these objects relative to the target object can be determined. Firstly, the orientation region containing potential facing objects is calculated. This is dependent upon the policy selected by the application. Each object is then examined in turn:

- If the object has no active sides it is denoted **unknown**.
- An object is denoted **facing** if the following two conditions hold:
 - 1. If at least one of the object's active sides intersects the orientation region.
 - 2. If at least one of these active sides lies on the same axis as the target object's active side and the direction of the active side is opposite to that of the target object's active side.
- If not yet denoted the object is now denoted **not-facing**.

This process is repeated for each active side of the target object. The result of this is that each object originally found in the spatial search now has an orientation associated with it, relative to the target object.

Registering a Spatial Query

To provide asynchronous query capability an application can register interest in a region of space. This is a powerful technique, enabling the location service to notify the application whenever an object is seen intersecting or contained within this space.

The application submits to the location service a rectangle describing the region of space, a containment/intersection flag, an optional event template and a callback handle. A unique identifier is returned to the application which it can use at a later time to de-register. A search is immediately performed within this space and the object records satisfying the search are returned to the application. In addition the location service creates a *search entity* consisting of a unique identifier and the search rectangle. The search entity is distinguished by its type. The search entity's rectangle is inserted into the R-tree.

These search entities are used in the following way. When an object's position is updated, the MBR representing its current location is firstly found. A search then takes place to find those search entities intersecting, or contained within this MBR. Similarly, after the MBR representing the new location has been inserted into the R-tree, a search is again performed to find those search entities intersecting or contained within this new MBR. Two lists of search entities now exist. The first is used to inform applications that the updated object has *left* the specified spatial area. The second is used to inform applications that the updated object has *entered* the specified spatial area. To prevent unnecessary call-backs the application associated with the search entity is only informed once if an object intersects a spatial area before *and* after an update. This search entity mechanism provides a simple and efficient way of implementing event-driven spatial searching.

Registration of a *proximity query* is also supported. The search entity is determined by the centre point of the named object, the error in the object's location and the application provided distance. This is then inserted into the R-tree and call-backs take place as described above. In addition the search entity is re-inserted whenever the location of its bound object is updated. The new search entity will then intersect or contain a new set of objects. These are determined and the application informed.

7.5.2 Name Queries

An application may wish to find the current location of a named object. It submits a name and type to the location service. This (name, type) pair is used as a key into the hash index previously described. The index returns a pointer to the object's record which is in turn returned to the application as the result.

Name Query Registration

An application may be interested in being informed whenever the location record for a named object is updated. It therefore registers with the location service and provides a call-back handle for the location service to store and later invoke. The current record for that object is returned along with a unique identifier which the application can use later to de-register. The call-back is then amended to a list of such call-backs attached to the object. When the object record is updated the call-backs associated with it are invoked with the contents of the new record.

7.5.3 Summary

Figure 7.12 summarises the relationship between object records, bound name queries, bound proximity queries, search entities, and the R-tree index.



Figure 7.12: Relationship of Indexed Object Records

7.6 Representing the Environment

The term *environment* is used to describe static parts of a typical spatial domain, such as walls, floors, doors, windows, stairs etc. The layout of buildings may change but does so infrequently.

By representing the environment, the functionality of the location service can be extended and further types of query answered such as:

- what objects are within room R?
- what is the distance between rooms R_1 and R_2 ?
- what objects are within 10m walking distance of object x?
- is the screen visible from location L?

Here, a method of representing rooms and the connections between these rooms is presented.

7.6.1 Representation of Rooms

Artificial human environments, for example homes and offices, are typically made up of a set of interconnected rooms. These may vary in size - some may be small, others may be considered "open-plan". However constructed, the design influences the context-aware environment in terms of the way people operate, particularly when interacting with a physical object.

Consider the following scenario, illustrated in figure 7.13.



Figure 7.13: Necessity for Clipping

Two rooms, A and B, and a corridor, C, are depicted. Person P is currently within room A. An application wishes to know those objects within a distance x of person P. The MBR approximation to this query is shown by the dashed rectangle. If this query is evaluated as previously described, objects r and s are both returned from the search.

At first sight this result is satisfactory. However the wall between rooms A and B prevents the objects P and s interacting. An application may quite reasonably expect the system to take account of this restriction. In fact the requirement that physical barriers are handled during the evaluation of a query is the general case for *physical* objects, as it can be assumed that physical interaction is the aim. *Non-physical* objects however, such as radio fields, are not inhibited by walls and therefore in this case no account of such physical barriers is required.

Clipping

A solution to this problem is to ensure the search is restricted to within the container formed by the room. *Clipping* is used. Clipping is a well known technique, widely used in computer graphics and windowing systems. The basic idea is that given two geometric shapes, one is *clipped* to the other. Typically the shape used to clip is a rectangle, as illustrated in figure 7.14.

If clipping is applied to the example in figure 7.13, then P's search extent will



Figure 7.14: Clipping in Practice

be clipped to the wall between A and B. The result of this query will not therefore include object s. Whether clipping is applied or not can be specified by the application when submitting a query.

Clipping can also be applied to object spatial extents. This ensures that an object with a positioning error, perhaps positioned close to a wall, does not appear within two containers at the same time. Clipping of an object's extent is applied when the position of the object is being updated, prior to MBR insertion into the R-tree index.

Clipping may add significantly to both the algorithmic and computational complexity of the system. However, as will be described, by restricting the representation of rooms to collections of rectangles, the resulting clipping algorithms can be made as efficient as possible.

7.6.2 Internal Representation

The spatial extent of a room is represented by a set of non-overlapping rectangles. This enables an efficient clipping algorithm to be used. In most cases a single rectangle can be used to represent a room. However in the case of a non-rectangular shaped room it can be approximated with more than one rectangle. Figure 7.15 shows an example decomposition.



Figure 7.15: Use of Rectangles to Approximate a Convex Polygon Shaped Room

The container rectangles are stored in a separate R-tree index within the location service. Containers are not stored in the main index, along with the object extents, as containers are not required to be returned when a spatial search is executed. A separate structure speeds up access to the containers.

When a query is submitted clipping can be specified. The query search extent may straddle several containers. To decide which container the extent should be clipped to, the extent's centre point is determined. It is then straightforward to determine which container holds this point. This container is then used to clip the search extent before searching takes place.

Where multiple rectangles represent a container clipping must take place against each one. This results in a search for each component rectangle. A simple naming scheme facilitates the cross-referencing of rectangle components. For example the centre point of the spatial extent may be contained within 'Au501:2;5'. 'Au501' indicates the container name (normally the name of the room). The suffix ':2;5' indicates that this is the second rectangle of five. A hash index allows the quick retrieval of other container components. Clipping and searching may then take place on each component. The objects returned from each search are grouped together.

7.6.3 Searching by Container Name

An application can specify a search, providing a container name as a key. The container is then found and a search within this takes place. In this way an application can ask "what objects are in this room?", which is convenient and intuitive.

7.6.4 Communication Between Containers

The representation of containers allows the clipping of search extents leading to better support for physical objects. This is an important result. However containers are rarely totally closed but are joined together. This communication is typically provided by a doorway.

Consider figure 7.16. This shows part of the CL, namely floor 5 of the Austin building. Floor 5 consists of two rooms, 501 and 502, and a short corridor that is termed 5-landing. Rooms 501 and 502 have doors leading to 5-landing. Three



Figure 7.16: Austin Building, Floor 5

containers are constructed, one for each room. Each container is represented by a single rectangle.

The unclipped query extent is shown by the dashed rectangle. As the centre of this extent is within 501, clipping reduces this to a search extent wholly within 501. The object A will therefore not be returned by the query evaluation. To enable queries to extend through room interconnections, a new spatial entity is introduced which represents these points of interconnection between one container and another.

Junctions

Room interconnections are represented by *junctions*. These are positioned at the borders between containers and are the only type of object that may simultaneously be within two or more containers at the same time. Figure 7.17 shows floor 5 with junctions added. In a three-dimensional system junctions



Figure 7.17: Floor 5 with Junctions

are two-dimensional. Typically they will be the width and height of a doorway but have no thickness.

Junctions are inserted into the main system R-tree at initialisation. When a query is executed a junction may be returned as a result. If the application has specified that searching should be extended, junctions are interpreted and a new search takes place in the adjoining container. The objects found from this new search are then grouped with those already found. The search continues until no new junctions are found.

This results in a search based upon "walking distance" and user accessibility. The search area is repeatedly reduced based upon the distance between the centre of the search area and the position of the junction.

Search Algorithm Detail

The junction search algorithm is given an initial search rectangle S. It operates as follows:

- 1. Determine container C which holds the centre point of S. Clip S to C resulting in $S_c.$
- 2. Search R-tree index for those objects that intersect/are contained within $S_c\,.\,$ Add objects found to result set.
- 3. For every junction J found in search:
 - (a) If J is a member of the set of junctions already seen, stop.
 - (b) Add J to junction set.
 - (c) Determine the centre point of intersection between ${\boldsymbol{J}}$ and ${\boldsymbol{S}}$.
 - (d) Construct new search rectangle S_n based upon S_c and the position of $J\,.$
 - (e) Determine containers that ${\boldsymbol{J}}$ intersects.
 - (f) Recursively invoke search algorithm with \boldsymbol{S}_n .

This algorithm is illustrated in figure 7.18.



Figure 7.18: Searching using Junctions

Rooms and corridors are depicted by solid lines, and search extents by dashed lines. The original search extent is labelled A. Its centre is at point 1. After clipping, the search is conducted and a junction is found, centred at point 2. r is the distance in one-dimension between points 1 and 2. s is the size of the original search extent in the same dimension. The size of the new search extent is calculated by subtracting r from s. This is done for each dimension. A search is then conducted using the resulting search extent, B. This is done in each of the containers in which the junction exists. In the second search, another unseen junction is found, centred at point 3. A new search extent is constructed, C, and another search is executed.

7.6.5 Environment Database

Information about containers and junctions is considered to be part of the static environment. As such, responsibility for maintaining this information will typically lie with a system administrator, and updated when, for example, a room is partitioned forming two smaller ones. Updates may be facilitated by the use of a graphical tool.

Two databases are maintained, both with CORBA interfaces, to allow the location service to initialise itself at start-up. Sample files which provide this support are shown in appendix B.

7.7 Application Interface

Location updates and synchronous and asynchronous search queries are accessed through CORBA interfaces. Here, three of these interfaces, defined using IDL, are described.

7.7.1 Update Interface

long update(in cuboid_object object_record, in long clipping_flag);

This interface is used by location sensor interfaces to update an object's position. cuboid_object is a composite data type consisting of:

- identifier the object's name;
- type the object's type;
- location consisting of two points together defining a cuboid. This represents, in three-dimensions, the MBR of the object's spatial entity;
- error a single point representing the maximum error in each dimension;
- time-stamp an integer representing the time at which the location sighting occurred.

clipping_flag indicates whether the object's MBR should be clipped prior to its index insertion.

7.7.2 Search by Name Interface

```
long location-of(
    in string identifier,
    in string type,
    out cuboid_object object_record,
    out string container_id
);
```

This interface (described in section 7.5.2) enables the return of the location of the object with name identifier and type. The latest record for the object is returned in object_record with also the name of the container the object is currently within.

7.7.3 Registration Interface

This interface (described in section 7.5.1) takes a search area and inserts a search entity into the R-tree. This search area is monitored and each time an object enters or leaves the search area the location service invokes the call-back provided by the application.

The interface takes as parameters:

- search_area a cuboid specifying the extent of the search object;
- clipping_flag should clipping to a container be applied to the search area;
- enter_leave_flag should the application be informed only when a new object enters or leaves the spatial area;
- contain_intersect_flag whether objects should just intersect or be wholly contained within the search area;
- orientation_area_flag how the spatial area to search for qualifying active sides is to be calculated;
- callback_object a reference to a CORBA object used to perform callbacks;
- current_objects this is an array (CORBA sequence) of objects intersecting the spatial area at the time of registration.

7.8 Summary

The design and implementation of a location service has been described. Each physical object is represented using a simple geometric shape, a threedimensional, axis-aligned, cuboid. This provides, in most cases, a sufficient approximation to the size and location of a real object and allows the use of simple, and therefore efficient, manipulation algorithms. This will ensure performance requirements are met. Approaches to implementation were described and a lightweight method adopted using an R-tree index. The process of query evaluation was then detailed. Two aspects of environment representation were also discussed and extensions made to the system to allow querying to take account of this representation. Finally aspects of the CORBA application interface were described.

The following chapter presents an evaluation of the location service in terms of functionality and performance.

Chapter 8

Location Service Evaluation

This chapter evaluates the location service, firstly by case-study to assess its capabilities, and then quantitatively by measuring its update and search performance.

8.1 The Location Service in Use

The location service should be capable of representing a realistic situation and supporting context-aware applications. Its ability to support the following will be shown:

- Location information integration: multiple sources of location information must be supported.
- *Environment representation:* the static environment must be adequately represented.
- Applications support: the asynchronous monitoring of spatial areas and other query facilities must be available.

Evaluation will consist of a worked case-study, typical of the targeted system domain. The rooms and corridors of floor 1 at ORL are encoded using containers and junctions. Three location systems are used to provide the positioning of static and mobile objects. Objects and rooms are represented in three dimensions.

8.1.1 Representing the Location Domain

Floor 1 of ORL is illustrated in figure 8.1, annotated with room names. This plan has been derived from original architectural plans. The coordinates of rooms are converted to metres and are relative to a fixed point on the building.



Figure 8.1: Floor 1 of ORL

Containers

Rooms are represented by containers, which are axis-aligned cuboids. Figure 8.2 shows the containers required to represent the rooms.

11001 1							R112		
	R107	R108	R109	R110	R111	Hall	R114	R113	
R107		Hall			Hall				
	R107a	R106	R105	R104	R103	R101			

Figure 8.2: Floor 1 Containers

Most of the rooms are represented by a single container. The exceptions are R107, which requires two containers and the *Hall* which requires three. A single container is used to represent the floor itself. For reference the container information for floor 1 is presented in appendix B.

Junctions

Junctions are used to represent the interconnections between rooms. In the case of floor 1, all rooms except one are connected to the *Hall*. The exception is room R107a which is connected to R107. Each room has a door and therefore the positions and dimensions of doors give the correct positions and dimensions of junctions. Junction positioning is shown, with containers, in figure 8.3.

The junction information for floor 1 is also presented in appendix B. This completes the representation of floor 1.



Figure 8.3:	Floor	1	Junctions	and	Containers
-------------	-------	---	-----------	-----	------------

8.1.2 Location Sources

Three sources of location information are used:

- Active Badges: the system is deployed throughout floor 1. Users and some types of equipment are equipped with Active Badges.
- **Ultrasonic Badges:** *R107* is partially covered by the ultrasound positioning system.
- Manual Measurement: positions of equipment which move rarely can be measured manually.

Any object has its location reported by at most one of the above systems. The interfaces of the system to the location service are now described.

Active Badge System



Figure 8.4: Floor 1 Infra-red Zones

Recall that each Active Badge receiver defines the extent of an *infra-red zone*. Most rooms can be represented with a single IR-zone. To cover a large, or irregularly shaped room, a number of infra-red zones are required. The approximate IR-zones formed on floor 1 are shown in figure 8.4 by dotted rectangles. These zones are approximated to cuboids for representation within the location service.

These IR-zones dictate the level of granularity of information obtained from the Active Badge system. The location service requires an input in the form of a cuboid, and error. The CALAIS Active Badge system interface derives a suitable position of an object from the infra-red zone that the object is seen within. If it is seen in more than one zone concurrently, an arbitrary choice of zone is made. The object's position is defined to be the centre of the zone. The appropriate cuboid is formed from this and the object's dimensions, assuming that the sensor is positioned at the centre of the object. The error is equal to half the size of the infra-red zone, minus the extent of the object. This is illustrated in figure 8.5.



Figure 8.5: Error Derived from an Infra-Red Zone

The dotted rectangle indicates the extent of the cuboid indexed, equivalent to the size of the infra-red zone. The location service therefore receives sightings of objects which are typically positioned at the centre of a room, but with an error implying the object may have been seen anywhere within that room. The cuboid which is indexed for each object has therefore the same dimensions as the infra-red zone it is seen within.



Figure 8.6: Active Badge Interface

Figure 8.6 shows schematically how the Active Badge system is interfaced to the location service.

Ultrasonic Positioning

The ultrasonic system is in prototype form and therefore only covers a small area of floor 1, specifically part of room R107. Nevertheless, because it provides fine-grain location information, it forms an important part of the evaluation.

Ultrasound badges are available for tagging people or equipment. They enable a three-dimensional position to be calculated. The cuboid expected by the location service is therefore created based upon this position and the size of the object. The badge location is again assumed to be at the centre of the object. The error is constant for each sighting and is 10cm in each dimension. Figure 8.7 shows the relationship between the location sighting, the dimensions of the object and the error.



Figure 8.7: Error Associated with an Ultrasound Badge Sighting

The dotted rectangle indicates the extent of the cuboid actually indexed.

Figure 8.8 shows how the ultrasound system is interfaced to the location service.



Figure 8.8: Ultrasonic Badge Interface

Manual Measurement

For evaluation purposes it is helpful to include the location of some objects which can neither be tagged by an Active Badge, or by using ultrasound. The positions of these are determined either by temporarily positioning an ultrasound badge next to the object, or by the use of a electronic tape measure. Orientation of some devices is also determined. The locations of objects determined in this manner are inserted into the location service once only, when it is first started. A location error of 10cm is used to allow for manual measurement errors.

8.1.3 Database Resources

Object Dimensions Database

A database is maintained holding details of object dimensions. A lookup is performed using a CORBA interface, by the location source interfaces, prior to an update to the location service. The object dimensions retrieved are used, as previously described, with the location position error, to determine the actual cuboid to be indexed for the object.

Object Capabilities Database

Another interface is provided to a database holding information concerning the capabilities of equipment. This enables an application to determine those types of equipment which are capable of delivering particular functionality. This database is simple in format and consists of a set of name, type and capability string pairs. The string encodes a set of capabilities delimited by commas. By using the interface to this database an application can determine, for example, if an item of equipment is capable of displaying a teleport session. The application submits the name and type of the item of equipment, and a string, such as 'teleport'. It is then determined whether this string matches a substring within the appropriate item's capability list and this result is then communicated to the application.

8.1.4 Query Evaluation

The aim of the location service is to provide support for context-aware applications. To illustrate its effectiveness a set of example applications are now presented. The operation of each application is described, together with the additional information resources required, such as the object capabilities database. The location service interface which the application uses in each case is indicated.

1. Where is object with name X and type Y?

Using the interface location-of, an application can specify a name and type of object and be returned the coordinates of the cuboid specifying its size and position, the container in which the cuboid is positioned, and the error in the cuboid's position. The location service searches by name and type and returns the time, location, error and container of the object's last sighting.
For example:

```
'bean,NT' was last seen at time 872270257 576272
location: (26.85, 19.1455, 1.35), (27.15, 19.4455, 1.65)
container: R101
error: 1.21364, 2.00909, 1.35
```

2. What are the names and locations of objects in room R?

A spatial query is required. The application must first find the coordinates of the containers that represent the room. It obtains these from the environment database described in section 7.6.5. It then uses the **spatial-search** interface to search each of the room containers in turn. The result might be:

```
2 objects returned
'parsnip,DDS' was last seen at time 872280547 849660
location: (13.7591, 25.125, 1.35), (14.0591, 25.425, 1.65)
error: 0.940909, 1.175, 1.35
'okra,solaris' was last seen at time 872280797 252324
location: (13.7591, 25.125, 1.35), (14.0591, 25.425, 1.65)
error: 0.940909, 1.175, 1.35
```

Such a query may be issued on a regular basis by a system administrator to maintain an inventory of equipment. By examining the time-stamps it could also be used to determine whether any people were currently using a room.

3. Keep me informed of the locations of objects of type 'person'.

The application registers interest in changes in location of the object type *person*, using interface event-location-of. This is done by providing an appropriate object template and a call-back interface reference. Each time an object of this type changes position, the application is informed. This is the asynchronous equivalent of example 1. Such a query might be used by a graphical viewer, displaying current locations of people.

Sample output may be:

```
'gjn,person' was last seen at time 872271001 676772
location: (26.85, 19.1455, 0.0), (27.15, 19.4455, 2.0)
container: R101
error: 0.1, 0.1, 0.1
```

then at a later time:

```
'jdp,person' was last seen at time 872271002 453234
location: (19.244, 18.96, 0.0), (19.744, 19.46, 2.0)
container: R111
error: 0.1, 0.1, 0.1
```

4. Where is the nearest object of type 'printer' to the current location of person 'gjn'?

This is a *relative proximity* query and uses the **proximity** interface. The application must specify an object name and type, and a spatial region. This will be in the form of an x, y and z extent relative to the current location of person 'gjn'. This will determine the maximum range over which the spatial query will be executed.

Additionally the application must indicate using binary flags whether the query should use clipping, junction traversal and whether the objects returned should be wholly contained within, or just be intersecting, the search area.

The nearest object can be found easily by the application as all the objects satisfying the query will be returned in an array, ordered by distance from the originating object 'gjn'. This type of query determines relative orientations, the target object being 'gjn'. However in this example no object active sides have been encoded, leading to all orientations being described as 'unknown'.

Figure 8.9 shows the positions of printers on floor 1. Consider first that



Figure 8.9: Printer Locations on Floor 1

the application is indiscriminate and wishes the query to cover most of the floor. An (x, y, z) region of $(\pm 15m, \pm 7m, \pm 3m)$ is chosen. Such a search region, centred at the position of 'gjn' is partially shown.

(a) With Clipping. The query returns the following:

```
'nuclear,printer' was last seen at time 872532618 142974
location: (4.5, 19.5182, 0.7), (4.9, 19.9182, 1.1)
container: R107;2
error: 0.1, 0.1, 0.1
orientation: unknown
```

As clipping is required, the centre of the search region is found. This is of course at the position of 'gjn'. The container enclosing 'gjn' is

90

then determined. The search region is clipped to this container and the search then executed. In this case in fact, the container enclosing 'gin' is one of the three (see figure 8.2) which represent room R107. This situation is recognised, the other containers determined, and a clipped search performed within each.

(b) Without Clipping. On this occasion the query returns the following:

```
'neutron,printer' was last seen at time 872534028 545454
location: (10.5, 19.56, 0.7), (10.9, 19.96, 1.1)
container: R108
error: 0.1, 0.1, 0.1
orientation: unknown
'nuclear,printer' was last seen at time 872532618 142974
location: (4.5, 19.5182, 0.7), (4.9, 19.9182, 1.1)
container: R107;2
error: 0.1, 0.1, 0.1
orientation: unknown
'phaser,printer' was last seen at time 872534026,175754
location: (24.0909, 23.3273, 0.5), (24.4909,23.7273,0.9)
container: F1HALL;2
error: 0.1, 0.1, 0.1
orientation: unknown
```

The printer 'neutron' is the nearest to 'gjn', indicated by it appearing first in the returned array of results.

(c) With Junction Traversal. Now consider if the application wishes to extend the search to other containers but retain clipping. In this case the application is interested in printers within approximately 15m traversal, or "walking distance", from 'gjn'. The following is returned:

```
'nuclear,printer' was last seen at time 872532618 142974
location: (4.5, 19.5182, 0.7), (4.9, 19.9182, 1.1)
container: R107;2
error: 0.1, 0.1, 0.1
orientation: unknown
'neutron,printer' was last seen at time 872534028 545454
location: (10.5, 19.56, 0.7), (10.9, 19.96, 1.1)
container: R108
error: 0.1, 0.1, 0.1
orientation: unknown
```

In this case the search, conducted using the algorithm described in section 7.6.4, does not extend far enough to encompass 'phaser'. This is because the traversal distance from 'gjn' to 'phaser' is further than the point to point distance.

'nuclear' is the nearest printer in this case as it appears first in the results array.

5. Inform me when object X is in room Y.

The application must give the location service a spatial region and a callback interface reference to be invoked whenever an object is seen in this region.

For example, consider if the application wishes to know when person 'gjn' is seen within the room R106. Firstly the R106 container is looked-up in the environment database. Then the region formed by this container and a call-back interface reference are given to the location service together with a template specifying the name and type of objects of interest.

Call-backs may occur as follows:

'gjn,person' was last seen at time 872544957 484152 location: (10.9182, 25.125, 0.0), (11.2182, 25.425, 1.80) container: R106 error: 0.872727, 1.175, 1.35

and another 10 seconds later:

'gjn,person' was last seen at time 872544967 518234 location: (10.9182, 25.125, 0.0), (11.2182, 25.425, 1.80) container: R106 error: 0.872727, 1.175, 1.35

8.1.5 An Extended Example - Promiscuous Teleporting

Context aware applications will often submit queries of the form:

Keep me informed of the objects within spatial region R, where R is defined relative to the position of object A.

For example, inform me of all NT workstations within 1m of person 'gjn'. In this case the location service is given an object name and type, a relative search extent of 1m on each axis, and a call-back interface reference. The interface event-proximity is used. Every time an object sighting within 1m of 'gjn' is reported, a call-back with the object sighting is performed. Also, each time the location of 'gjn' is reported, the record for each object within 1m of the new location is reported using a call-back. Relative orientations will also be given expressed in the manner described in section 7.4.1.

This interface is used to build an application termed *promiscuous teleporting*. Recall the teleport system. A user can move their X-based applications between displays. This is done manually or under control of an Active Badge. As an extension of this, a user may have a number of applications which they wish constant access to. Examples may include an e-mail reader, a Web browser, an active diary and a weather indicator. The user would like such applications to 'pro-actively' follow them wherever they are and be displayed on the nearest appropriate device. The performance of the teleport system is suitable for supporting such a mobile application, typically taking less than 5 seconds for materialisation to occur. This chapter discusses the location requirements of such applications. Chapter 9 shows how further requirements for this application can be satisfied.

Basic Semantics

The basic semantics of promiscuous teleporting can be described as follows:

Whenever the user is within X metres of equipment capable of accepting a teleport session, materialise their promiscuous teleporting session on the nearest display.

When no such display is available, de-materialise the teleporting session, if it is not already de-materialised.

Each user has a promiscuous teleporting *agent* which uses the location service to discover devices close to the user. It must then find out whether any of these devices are able to accept a teleport session by using the equipment capability database. Once a suitable device has been found, teleporting takes place. The nearest device is used. If two devices are at the same distance, one is arbitrarily chosen. If the orientation of a device is known, this is also used to determine its suitability. If no such devices exist within the distance required, the teleport session disappears from view until a device close enough is available.

How it Works

The agent firstly registers interest with the location service in location events occurring within Xm of the user. For example, 2m of user 'gjn'. Most of floor 1 is not covered by ultrasound and it is desirable that the system is operable over the whole floor. Taking this into account a maximum error of $\pm 3m$ is specified. Active Badge information may therefore be used. Equipment must be within the same room as the user, therefore clipping is enabled. Junction traversal is not enabled as 'line-of-sight' proximity is required. Orientation information is also returned, and where the orientation is known this can be used to discriminate between equipment.

All equipment within the range from the user is reported. To find out whether the equipment is capable of teleporting, the object capability database is consulted. In this case the capability submitted will be the text string 'teleport'.

More than one piece of equipment, capable of displaying a teleport session, may be returned by a call-back. The items in the call-back are sorted by proximity the first one in the list is the nearest, and therefore by examining the list from beginning to end and choosing the first piece of equipment with the required capability, the nearest piece of suitable equipment will always be chosen.

Each time the location of 'gjn' is updated, a call-back occurs. Only those objects newly intersecting the derived search extent are reported. Similarly, when an object is no longer intersecting this spatial extent, a call-back is performed to indicate this.

Promiscuous Teleporting in Operation

Consider the equipment setup illustrated in figure 8.10. The extent of the



Figure 8.10: Equipment Capable of Accepting a Teleport Session

ultrasound system is shown by the dotted rectangle.

'dasheen', 'chestnut' and 'abalone' are workstation displays which can accept teleport sessions. 'dasheen' and 'chestnut' are monitored using ultrasound. Furthermore their orientation is also encoded, although for simplicity this is done manually and not using multiple ultrasound devices. Each of these displays has a single active side, shown in figure 8.10 by a bold-line.

'abalone' is monitored using a badge. 'arling' is a *video-tile*, a dumb graphics terminal with a pen I/O device which can also accept a teleport session. 'gjn' is monitored using an ultrasound badge within R107, and an Active Badge elsewhere. 'gjn' has all six sides denoted as active, thereby allowing relative orientation of 'chestnut' and 'dasheen' to be determined.

The user, 'gjn', is moving from office to office on floor 1. As 'gjn' enters the proximity of any of the above devices his teleport session is materialised. For 'abalone' and 'arling', this occurs when 'gjn' enters R107a or R112, as both devices are monitored using badges. In R107, fine-grain location information is available. Also the orientations of 'chestnut' and 'dasheen' have been encoded. As 'gjn' moves around the room he will be at times closer to 'chestnut' or 'dasheen'. Due to the 3m error threshold, both display names will be reported back but will be ordered in terms of distance from 'gjn'. Furthermore relative orientation information will be returned. In the cases where such information is available and where the display is indicated to be facing, the teleporting session

will be materialised. This will occur, however 'gjn' is oriented, as all six sides of the representative cuboid are denoted active. In cases where orientation is indicated to be *not-facing*, the session is de-materialised. The shaded area in figure 8.10 indicates a region which is greater than 3m from either workstation. When 'gjn' is within this region the teleport session is also de-materialised.

8.1.6 Summary

This qualitative evaluation section has shown that the location service is capable of delivering rich functionality to applications. The following capabilities have been used:

- Representation of one floor of a typical office domain.
- Integration of multiple sources of location information.
- Querying by object name and type.
- Querying by spatial region.
- Querying by spatial region, determined relative to a named object.
- Queries using clipping and junctions, enabling account to be made of the static environment.
- Asynchronous monitoring of spatial regions.

8.2 Location Service Indexing Performance

The performance aims of the location service were previously described in section 6.2.5. These can be summarised as follows:

- 500 updates performed per second;
- 2000 queries and call-back search entities held.

Evaluation of location service performance is presented in two parts. The Rtree index is a key factor in determining performance. It is therefore examined in isolation. Secondly, the location service is evaluated as a whole.

To briefly summarise, an R-tree is an $O(\log n)$ structure, used, in three dimensions, to index axis-aligned cuboids. It is a derivative of the B-tree and thus taken account into its design is the ability to operate efficiently with a secondary storage medium such as disk. The implementation within CALAIS is memory-resident. Experiments were conducted on a Sun UltraSparc, 168 MHz machine with 4Gb of main memory. Code was analysed using a profiling tool and where possible optimisations then performed. Timings were determined using a Unix profiling timer, which measures the CPU time taken by a process and by the system on behalf of the process. Such a time is therefore independent of other load on the machine. Each experiment was conducted 10 times to allow standard deviations, plotted as error-bars, to be calculated.

8.2.1 Determining the Size of an Indexing Node

Consider the way in which an R-tree works. Each internal node holds a number of records, n, making the tree an n-ary structure, where n is normally determined by the size of the internal record structure and the operating system's page-size. This is to ensure that when a node is accessed, each record within that node is read into memory from secondary storage for maximum efficiency. n determines the height H of the tree. $H = log_n(x)$ where x is the number of objects indexed.

In a memory resident system paging is not an issue. However an appropriate value for n must still be determined. Consider the effect n has on the functioning of the index. For a fixed number of records, as n increases in size, the height of the tree decreases and vice versa. As navigation through the tree takes place, entries within each node are examined sequentially. Thus as n increases in size, the amount of computation at each node to determine which subtree to navigate next is increased, and the amount of recursion in descending the tree is decreased. Thus the value of n is a trade-off between recursion and intra-node computation.

The value of n which is most suitable will depend upon the particular machine architecture and the efficiency of its run-time system and language compilers. To determine n for the target architecture, a number of objects, with randomly chosen extents, are indexed. 1000 updates are then performed, and a mean update time calculated. This task is performed for a range of values of n. The results obtained are shown in figure 8.11. This shows the results with 1000



Figure 8.11: Effect of Node Size on Update Performance

and 10000 objects in the index. Other index sizes were used but not illustrated for clarity. In all cases the minimum update time was achieved when n = 5. Similar results were obtained on a DEC Alpha machine running Digital Unix. The performance of the R-tree in this case was maximised with n = 6. This result confirms that assertion that the optimal value of n is dependent upon machine architecture.

All further results use the Sun UltraSparc machine and therefore a node-size of 5 is adopted for the remainder of this evaluation.

8.2.2 Object Density

Object density is defined as the number of objects intersecting a unit of space. Consider, in three dimensions, if a $4 \times 4 \times 4$ coordinate system is indexed and each object within it has unit size $1 \times 1 \times 1$. If 10 objects are present, each randomly positioned, the mean density will be $\frac{10}{64} = 0.156$. In general the object density is given by:

 $object density = \frac{number of unit sized objects}{coordinate system volume}$

As the number of objects increases, the density increases. As the coordinate system size increases, the density decreases. The density may range from 0 to I, where I is the number of objects indexed.

The object density will have an effect on the performance of the index. When conducting a search, a cuboid is used as a search extent and all objects found which intersect (or are contained within) this returned as a result. As the object density increases, more objects will intersect any given search area. More branches of the tree will therefore require descending, and the search will take a longer time.

Now consider updates which involve a delete and insert. Recall that each interior node holds a cuboid which contains those cuboids held in its subtree. When searching for the record to delete, the currently indexed cuboid is used together with the containment operator. A single unique path will therefore be traversed, irrespective of the object density. The cost of the succeeding insert is only dependent upon the height of the tree. Therefore update performance will not be affected by the object density.

To justify these arguments some experiments are conducted. 1000 randomly selected unit sized objects are indexed. 10000 searches and then updates are performed. This experiment is repeated for varying object densities. The results are shown in figure 8.12. The graph clearly shows that as the object density increases, the search time also increases. However changes in object density have, as predicted, no effect on update performance.

It can be concluded from these results that the density of indexed objects has a significant effect on searching performance. To attempt to calculate the object density within a deployed system, the estimate of 2000 objects within ORL



Figure 8.12: Effect of Object Density on Update and Search Performance

can be used. Assume each is on average 0.5m cubed. The dimensions of ORL are approximately $30m \times 14m \times 12m = 5040m^3$. The mean object density is therefore 0.4 objects per m^3 . This object density figure will be used in the next parts of the evaluation.

8.2.3 Index Size



Figure 8.13: Effect of Index Size on Update and Search Performance

The more objects are indexed, the greater the time taken to perform a search or update. A constant object density implies that a large total spatial volume is being indexed. Figure 8.13 shows how search and update performance alters as the number of objects indexed increases. Unit sized objects and search areas are used, with an object density of 0.4. As expected the time taken for a search and update increases as the index size is increased.

The R-tree index shows itself able to easily cope with large numbers of objects. With 4096 objects, approximately 5000 updates or searches per second can be performed.

8.2.4 Searching Performance

The final graph shown in figure 8.14 shows the relationship between search volume as a percentage of the total coordinate system volume, and search speed.



Figure 8.14: Search Performance

At 100% every object indexed will be found by the search. At 50% approximately half will be found. Two plots are shown, one with an index size of 1000, the other with an index size of 5000. The results fit with what may be expected. In doubling the search volume the number of objects returned in the search will approximately double. Not only will tree traversal time increase, but also the time taken to assemble the resulting objects for the calling application. Furthermore search times for an index size of 5000 are approximately five times that for an index of size 1000.

Consider one of the ORL floor plans illustrated in section 8.1. Assume most queries will be clipped to room containers. An estimate can then be made of a typical maximum search percentage. Room R107 takes up approximately one quarter of floor 1 and therefore $\frac{1}{12}$ of the whole building. The resulting volume is $\approx 8\%$. With such a search volume and an index containing 2000 objects, approximately 3000 searches per second can be performed.

8.2.5 Summary

This section has provided an evaluation of the R-tree index used within the location service. It has shown that indexing performance is dependent upon the number of objects indexed, the density of those objects within the coordinate system, and the size of the volumes typically searched.

8.3 Location Service Performance

This section will examine the performance of the location service as a whole. The R-tree index is the heart of the system. However, the location service is responsible for handling the registration and de-registration of queries, indexing objects by their name as well as spatially, and performing event-driven queries. All this is accessed by clients using CORBA interfaces. Performance will therefore be lower than the raw performance of the R-tree.

Clients performing operations on the location service may be located on the same machine, or on a different machine communicating with the service over a network. Evaluating the speed of a network is not of concern here and therefore clients in this evaluation, whether performing an update or search, are located on the same machine and therefore can communicate with the location service using inter-process communication.

8.3.1 Updating

Updating of physical object positions is the basic function the location service must support. Objects are updated by *name*. A name and type are submitted with the new position. To update the index record associated with the object, the system must delete the old record. For this the old position is required. Firstly therefore, a search within the hash index is performed using the name and type pair as the key. The record found is then used to perform a deletion from the R-tree. The new position for the object is then inserted.

After the update has been performed, a search is executed to determine whether any search entities intersect the object just inserted. If so, a call-back needs to be performed.

To summarise, an update consists of the following operations:

- 1. Searching the hash index.
- 2. Deletion of old record from the R-tree.
- 3. Insertion of new record into the R-tree.
- 4. Search for search entities.

Figure 8.15 shows update and search performance as the number of objects stored is altered. As before, an object density of 0.4 is used. Each object is of unit size.



Figure 8.15: Location Service Update and Search Performance

Search performance is considerably higher than update performance. This is necessarily so as a search is a single operation compared to the four operations required for an update, one of which is a search. Compare the update performance for 4096 objects shown in figure 8.15 to that shown in figure 8.13 which represents raw R-tree performance. The figures are $\approx 0.5ms$ and $\approx 0.2ms$ respectively. The additional cost of searching and updating the hash index and performing a post-insertion search, as well as the CORBA interface overhead, accounts for the difference in time.

8.3.2 Evaluating Queries

Figure 8.16 shows the time taken to perform an update with a set of unit sized search objects registered with the location service. This illustrates the update performance of the location service when, in the post-update search, search objects are encountered and an application call-back is therefore required. Each call-back executed here is terminated at the point at which communication takes place with the appropriate application. Network communication cost is therefore ignored.

The number of physical object extents is fixed at 4096. With an increasing number of search extents registered, it it more likely that a post-update search will encounter one or more search objects, thus causing a call-back to occur. With 4096 search extents registered each update will trigger, on average, one call-back.

Over 1000 updates can be performed per second, with 4096 search extents registered. This result demonstrates that event-driven query evaluation, a fundamental aspect of the location service, is cheap to perform.



Figure 8.16: Update Performance with Registered Search Entities

8.3.3 Performance Conclusions

Recall that chapter 6 identified the performance requirements of the location service, when handling 2000 objects, to be:

- 500 updates per second;
- several thousand query evaluations per second.

The substantive conclusion which can be drawn from figures 8.15 and 8.16 is that the location service implementation meets, and exceeds, these requirements.

Chapter 9

Composite Event Management

This chapter presents a client-based approach allowing applications to more easily handle communication with event services. Motivations are firstly discussed, and a language suitable for the expression of *composite events* is proposed. This is followed by a discussion of which individual event occurrences should be used in the evaluation of a composite event. Methods of examining event state and the integration of time events are also discussed. Aspects of an implementation are described which proposes solutions associated with the distributed event model.

Following this, a qualitative evaluation is made of the composite system proposed. Performance indications are also made.

9.1 Motivations

A composite event is an asynchronous occurrence which is triggered by the occurrence of multiple simple or composite events.

Many applications require access to two or more event sources. Consider an application which wishes to execute an action when it receives an Active Badge sighting and a workstation activity event for the same user, within a time period of five seconds. The application must create one call-back interface for each of the two event services. When call-backs occur, communication must take place between the two call-back interfaces to ascertain whether both events have occurred within the time limit.

The observation here is that to implement a conceptually simple application is non-trivial. Also, the handling of multiple distributed event sources raises a number of other issues including the naming of event services, distributed time and network delays. Each application should not be expected to handle these issues independently and this leads to the need for a generic set of composite event services.

9.2 Requirements

A system suitable for the handling and composition of events from multiple sources must include the following facilities:

- *Ease of use*: an application's behaviour may be dependent upon two or more event sources. Building an application using conventional means is complex. Straightforward methods of registration, call-back and the interrogation of event class attributes should be provided. Expressing a complex composite event should be made simple.
- *State lookup*: each CALAIS event service stores state. Many applications need access to this information, therefore integration of state lookups is required. This facility can also be used to access database resources.
- *Timers*: the relative time that two events occur is important to many applications. Interest may be expressed in knowing when an event *a*, then an event *b* occurs, but only if *b* occurs within time *t* of the occurrence of *a*. Provision must therefore be made for *time events*.
- *Client or server based*: a composite event system should be available as a library, to be linked with an application at compile time. In this way the computation associated with composite event evaluation is distributed. In addition, a composite event service should be available to those applications which run on unsupported architectures, or do not have access to the appropriate composite event library.
- Distribution issues: event services typically run on multi-tasking machines connected to a local area network. A client of an event service is therefore exposed to processing and network delays. Such delays may result in events from different services arriving at a client in an incorrect temporal order. Furthermore difficulties are raised in producing a temporal order as each machine has its own clock. Solutions to these problems need to be provided.

9.3 A Language for Expressing Composite Events

A convenient way for applications to use event facilities and express event compositions, is to use a declarative language. This section proposes the basic elements of such a language.

An example of a composite event (CE) expression is as follows:

badgeclick -> ABadge(A, B, C, D, E, 'SIDE');

This expression is given the name 'badgeclick' and consists of the single event class 'ABadge', corresponding to the CALAIS Active Badge event service. This event class takes six parameters. In this expression five parameters are variable names and the sixth a literal string. Generally an event class has the form $N(a_1, \ldots, a_n)$ where N is the name of the event class and a_1, \ldots, a_n are its formal parameters where $n \ge 0$. The arrow, \neg , is a *follow-by* operator and its semantics will be explained below. The semi-colon terminates the expression.

If the evaluation of the CE expression is satisfied, the evaluation is said to *accept*. The result of the evaluation is the set of events which caused the CE to accept. This is termed the *event path*.

9.3.1 Basic Operators

Basic operators of the CE language are now described. In the initial examples, all event classes are shown with no arguments. If a is an event class, t_a denotes the time at which the event of class a occurred.

Follow-by

The follow-by operator is the most fundamental and important operator for building composite event expressions. Consider the following:

eg1 -> a() -> b();

This expression is named eg1 and comprises a followed-by b. Follow-by expresses a temporal relationship. The CE eg1 will occur if an event of class a and an event of class b occur where $t_b \ge t_a$. Other classes of event are not considered and play no part in the evaluation.

Each CE expression starts with a follow-by operator. By definition, the expression starts evaluation at time zero, and thus, in the above example, a() will always match on the first event of class *a* encountered. Such syntax allows control over how events are processed, as will be explained further in section 9.4.

Disjunction

Disjunction can alternatively be termed *exclusive or*. It is denoted by a pipe, |. For example:

eg2 -> a() | b();

eg2 will occur if an event *a* or an event *b* occurs. The result of an accepting evaluation will be the event which occurs first.

Conjunction

Conjunction can also be termed *and*. Conjunction is denoted by a caret, ^. For example:

eg3 -> a() ^ b();

eg3 will occur when both events of class a and b have occurred. a may occur before b or vice versa. Conjunction can alternatively be expressed using disjunction and follow-by:

```
eg3 \rightarrow a() \uparrow b() \equiv eg3 \rightarrow (a() \rightarrow b()) \mid (b() \rightarrow a())
```

Without

The *without* construct is used to terminate a CE evaluation if a particular event occurs. Without is denoted with a hyphen, -. For example:

eg4 -> a() -> b() - c();

eg4 will occur if an a is seen followed-by a b with no intervening occurrence of a c, that is when $t_a \leq t_b < t_c$. If an event of class c is seen between the occurrence of a and b the evaluation is said to reject.

9.3.2 Operator Precedence

The precedence of operators within the CE language are as follows, highest precedence shown first. Those grouped together have equal precedence and are evaluated from left to right.

| (disjunction), ^ (conjunction)
- (without)
-> (follow-by)

Parentheses can be used to overcome operator precedence.

9.3.3 Variable Instantiation and Matching

Each event class has a set of attributes. Recall that a client, when registering with an event service, can submit an event template. The service then communicates to the client only those events which match the template. The template can include null attributes which act as wild-cards and which therefore match against any value of an attribute.

A CE system must include facilities to enable an application to gain access to these attributes. The solution is to allow the use of variables and literals within CE expressions.

106

Consider again the 'badgeclick' example given previously:

badgeclick -> ABadge(A, B, C, D, E, 'SIDE');

The event class ABadge has six attributes. Five of these attributes are given variable names, A to E. The sixth attribute is a literal, a string enclosed within single quotes. The variable names are initially uninstantiated and are equivalent to a null value. Any attribute values are therefore acceptable, which instantiate the variables. The sixth attribute is a literal and must match the value of the event attribute. Evaluation continues until a badge event occurs which matches that specified.

Variable Matching

Consider now the following CE expression, which monitors for user activity from an Active Badge followed-by activity from the same user at a workstation:

```
useractivity -> ABadge(A, P, C, D, E, F) ->
WSactivity(X, Y, P);
```

Here, the variable P is of interest, appearing as the second parameter of ABadgeand the third parameter of WSactivity. For each of these event classes, Prepresents the user-id of the person causing the event. This CE expression firstly matches upon a badge event from any user. It will then match upon a workstation activity event, but only if the now instantiated contents of variable P match with the user-id produced from WSactivity. Variables are scoped over the entire CE expression.

The use of variables in this way offers a useful way of correlating between different events, reducing the number of CEs that are triggered and thereby reducing the burden on the application to analyse event paths after an evaluation has accepted.

An Example

 $eg5 \rightarrow a(X) \rightarrow b(Y) | c(Y) - d(X);$

This means A, followed-by B or C, without an intervening D. This expression is correct in terms of operator precedence but parentheses can be used for clarity reasons. Using parentheses the above becomes:

 $eg5 \rightarrow a(X) \rightarrow ((b(Y) | c(Y)) - d(X));$

All the event classes used in the expression take variables as parameters. At the beginning of evaluation all variables are uninstantiated. When an event of class a occurs, the variable X is instantiated with the corresponding attribute. This attribute value is used in the monitoring of event class d. Concurrently event classes b and c are monitored. The attribute of the one occurring first, so causing the expression to accept, will instantiate the variable Y.

9.4 Parameter Contexts

Parameter contexts were first described in [CM93] and the concept refined in [CKAK94]. The aim is to overcome a problem with conventional composite event detection. Consider the following CE expression:

eg6 -> a() -> b();

eg6 occurs when an event of class a is followed-by an event of class b. Consider now the following *event stream*, where each succeeding event occurs at a later time than its written predecessor. n_i denotes the *i*th event of class n:

$$a_1, a_2, b_1, a_3, b_2, a_4, a_5, b_3, b_4$$

Two *a* events, a_1, a_2 , are seen followed-by a *b* event. At this point the evaluation accepts. A question is raised: which event of class *a* should be used in the composition? Put more generally, which component events of an event stream should form the parameters (or event path) of a composite event? A flexible strategy is proposed here which allows the application itself to specify a method for the evaluation of events and then enables the interrogation of the resulting event paths.

In [CKAK94], five evaluation methods are identified, termed *contexts*. A brief description of these differences will suffice here. In the most general case, termed the *unrestricted* context, every possible event combination is created. The above example would produce (a_1, b_1) , (a_2, b_1) , (a_1, b_2) , (a_2, b_2) , (a_3, b_2) , (a_1, b_3) , (a_2, b_3) , (a_3, b_3) , (a_4, b_3) , (a_5, b_3) , (a_1, b_4) , (a_2, b_4) , (a_3, b_4) , (a_4, b_4) , (a_5, b_5) ; in fact all instances of $\mathbf{a}() \rightarrow \mathbf{b}()$ where $t_a \leq t_b$.

Contexts are introduced to reduce this large number of accepting evaluations. A context describes which events should be used as constituents of a composite event. If the *recent* context is used, three composite events would occur, with the event pairs (a_2, b_1) , (a_3, b_2) , (a_5, b_3) . In contrast, the *chronicle* context produces the event pairs (a_1, b_1) , (a_3, b_2) , (a_4, b_3) . The difference between the two is that the recent context overwrites an event with a more recent one of the same class, whereas the chronicle context uses the first event seen of each class. Other types of context have also been described which provide additional alternatives.

Applications are inevitably restricted if a single, or set of pre-defined parameter contexts is used. To enable flexibility another follow-by operator is introduced.

An Alternative Follow-by Operator

By default, the CE system described here uses the *chronicle* context. To add the flexibility required another follow-by operator is introduced, denoted by =>. This allows multiple instances of the same CE expression to be evaluated concurrently. To explain this process a set of variations of expression *eg6* will be used together with the event stream shown above. Consider firstly eg6 itself which uses the -> operator only. The events forming the resulting composite events can be expressed as follows:

eg6 -> a() -> b() results in: (a_1, b_1) (a_3, b_2) (a_4, b_3)

This shows that three composite events result from the event stream. The use of the -> operator allows only a single evaluation of a particular expression to be active at any one time. Only when an evaluation finishes can another start.

Now consider the following example which introduces the => follow-by operator:

eg7 => a() -> b() results in:

$$(a_1, b_1)$$

 (a_2, b_1)
 (a_3, b_2)
 (a_4, b_3)

The => operator allows more than one evaluation of the CE expression to be active concurrently. Evaluation proceeds in the following way. a_1 occurs and the first evaluation thread then monitors for an event of class b. Then, a_2 occurs. The => operator allows another evaluation thread to start. Two evaluations are now active concurrently, both monitoring for an event of class b. b_1 occurs and is used in each of the active evaluations. Two composite events occur with the event pairs (a_1, b_1) and (a_2, b_1) . Another evaluation starts when a_3 is seen.



Figure 9.1: Evaluation of CE Expression

This can more compactly be expressed with a state diagram expressing stages of evaluation, as shown in figure 9.1. This illustrates graphically how the evaluation progresses. When an event occurs, evaluation moves from one state to another on a follow-by arc (labelled with the event that occurred). Multiple => operators may originate from a single state. Only one -> operator may originate from a single state. Consider the next example, $eg8 \rightarrow a() \Rightarrow b()$. This produces four evaluations:



eg8 -> a() => b() results in: (a_1, b_2) (a_1, b_3) (a_1, b_4)

This evaluation will never terminate, but will repeatedly accept when an event of class b is seen.

The final variation is eg9 => a_1 => b_1 . This produces 13 accepting evaluations, forming the *unrestricted* context.

$$eg9 \implies a() \implies b() \text{ results in:} (a_1, b_2)(a_1, b_3)(a_1, b_4)(a_2, b_1)(a_2, b_2)(a_2, b_3)(a_2, b_4)(a_3, b_2)(a_3, b_4)(a_4, b_3)(a_4, b_4)$$

These examples demonstrate that the appropriate use of the two follow-by operators allow the application flexible control over which events are used in the evaluation of a composite event.

Conjunction Operator

The conjunction operator, ^, can be considered a macro as it is defined in terms of follow-by and the disjunction operator. This expansion should be altered, depending upon the follow-by operator used prior to the conjunction operands.

Therefore, for the => follow-by operator:

 $eg10 \Rightarrow a() \ b() \equiv eg10 \Rightarrow (a() \Rightarrow b()) | (b() \Rightarrow a())$

This ensures the appropriate parameter context semantics are retained.

9.5 State Lookups

As described in section 5.3.3, each CALAIS event service stores a limited amount of state regarding events it processes. It may not be possible, or suitable, for an application to wait for the next event of a particular class, and therefore the ability to query this state is integrated into the CE language.

By prefixing an event class name with the operator *state*, the event monitoring request is modified into a state lookup. For example:

eg11 -> a(X) -> state b(X);

The evaluation will wait for an event of class a to occur. It will then immediately lookup the most recent event of class b. This lookup uses the same attribute and parameter matching rules as for events. If the lookup satisfies the attributes given, the evaluation accepts (as in the above example) or proceeds to the next stage of evaluation. As **state** is used, the evaluation does not have to satisfy the conventional time predicate, that $t_b \geq ta$.

state c(A,B) where both parameters are initially uninstantiated, will return the most recent event of class c with the variables instantiated. state c('xyz', B), where one or more parameters are instantiated, will return the most recent event which has matching attributes.

Although a state lookup can be used to replace any conventional event monitoring request, the semantics of the evaluation are altered. If the lookup is not satisfied the CE evaluation rejects, rather than remaining active until the event expression is satisfied.

Accessing Databases

State lookups can be used to access static, non-event driven database resources. The database must have an appropriate interface providing the standard state lookup facilities required by event services. This provides a simple and convenient way to access databases from within a CE expression.

9.6 Timers

As discussed previously, time is an important element in the management of events. The time at which each constituent event occurs is part of the result of each composite event. The application is therefore free to inspect these time results and reject those composite event results which do not satisfy timing requirements. However it is convenient if time constraints can be expressed within a composite event expression.

To derive the types of timer required, state diagrams are used. These are illustrated in figure 9.2, where a represents a simple or composite event.



Figure 9.2: Types of Timer

1. There has been no a for time t.

If the timer expires, evaluation proceeds. If an event of class a occurs, the timer is reset and another a is monitored for. The language notation used to represent this is a(),t

- 2. This provides no extra functionality and is equivalent to simply monitoring for *a*. No language support is therefore provided.
- 3. An a must occur within time t.
 If the timer expires, evaluation of the monitoring thread rejects, indicated by the reject state. Language notation: a() < t
- 4. An a must not occur within time t.
 If an a occurs before the timer expires, the monitoring thread evaluation rejects. Language notation: a() > t

These timer primitives can be used with simple or composite events. In the case of a composite event the timer is initialised when the monitoring for the first constituent simple event begins. For example, the expression $(a \rightarrow b) < t$ will accept only if an *a* followed-by a *b* occurs within time *t*.

113

9.6.1 Use of Timers with State Lookups

The semantics of timers change when used in conjunction with state lookups. For example, the expression state a() < t will accept only if a matching event occurred less than time t in the past. Therefore instead of the time constraint being applied to a future occurrence it is applied to a past occurrence. The use of an expression such as state a(),t is not permitted as the semantics of this timer only make sense when dealing with future events.

9.7 Operating Within a Distributed Environment

9.7.1 Distributed Time

Recall from chapter 5 that distributed CALAIS components are synchronised, using NTP, to within 10ms. A description of an approximated time base[Kop92] is also made use of here. This assumes that all clocks are synchronised to within a time granularity Π . With this knowledge and the assumption that each clock has a granularity smaller than Π , two events can be ordered if they occur 2 Π or further apart. If they occur less than this they are considered *concurrent*. A more detailed discussion of this issue is presented in [Sch96]. With clocks synchronised to within 10ms, this is the value adopted for Π .

In some applications where event rates are very high and it is common that two events arrive within 2Π , or 20ms, of each other, difficulties would be caused with the evaluation of temporal CE expressions. It is envisaged that this will not be common in the domain considered here. To enable correct treatment, the application is informed within its call-back if the given event ordering may be incorrect. The application can then decide on what action to take.

9.7.2 Network Delays

Consider two events, A and B, with time-stamps t_A and t_B where $t_A < t_B$. Due to network delays t_B may arrive before t_A . Let the expression being monitored for be A - B. In this case the expression will incorrectly reject, as B arrives before A.

Various approaches have been taken to solve this problem. In [SR90] buffering of events is performed for a time t_{max} , where t_{max} is the maximum anticipated delay throughout the system. A heartbeat protocol is described in [Hay96] which consists of servers periodically informing clients of pending events. Here an application oriented approach is taken. When a CE expression is submitted for evaluation, an additional parameter can be given which specifies the length of time events should be buffered, to allow re-ordering, before being consumed in the expression evaluation. An application that has no requirement that a correct temporal ordering is maintained can specify a delay of zero. In this case events will be consumed immediately. An application that is concerned with correctness will specify a buffering delay high enough to allow appropriate re-ordering but low enough to maintain adequate system response.

9.7.3 Registration Delays

The CE system must register with the services that provide the events specified in the CE expression. There are a number of possible registration strategies. Consider the expression $a() \rightarrow b()$. One approach is to register on demand with each service. Registration for event class b will take place after an event of class a has occurred. Registration incurs a delay and therefore it is possible that an event of class b will occur between the occurrence of a and the registration in b. It will therefore be missed.

One solution, proposed in [Hay96], is a pre-registration approach. A client informs a service that it wishes to receive events from the service in the future. The service then buffers events until the client registers again. This approach ensures events are not lost, but requires additional complexity in the event server and also requires the client to register twice.

The approach taken here is to immediately register interest in all event services contained within the CE expression. Using the example above, many b events will be received by the client. After the consumption buffering delay these can be disposed of. This approach increases the number of events communicated to clients but ensures no events are lost, and also simplifies the construction of both services and clients.

9.8 Implementation Considerations

9.8.1 Application Interface

An application may use the CE system either by communicating with a service, or by using a CE library within the application process. Both are equivalent in terms of functionality.

In a similar way to other CALAIS components, the CE system uses CORBA interfaces. The location transparency of CORBA objects mean that the application can use the same interfaces, whether using the CE system as a library, or as a remote service. This method also allows the CE system to be used with any application language capable of communicating with CORBA.

The application submits an expression as a string, together with a reference to a call-back object. The CE system uses the call-back object to notify the application when the composite event occurs. Also passed in the call-back is a data structure which holds details of the class names, associated parameters, and time-stamps of the events which caused the CE to accept. This *event path* can be interrogated by the client. This process is illustrated in figure 9.3.

Typically the contents of the event path will be used by the application to determine its succeeding behaviour.



Figure 9.3: Submission of CE Expression

9.8.2 Initialisation

Whether the composite event system is operating as part of a client process or as a service, initialisation is required before it can accept a composite event expression for evaluation.

The initialisation process interrogates the event class repository, described previously in section 5.3.4. From this it receives a description of event classes that have been registered with the system. This includes the name of the event class, a description of its attributes and an interface reference. This information enables a submitted expression to be parsed and error checked, and the appropriate services to be contacted.

9.8.3 CE Monitoring

Once initialisation is complete, the CE system begins the monitoring of incoming events. This monitoring system, illustrated in figure 9.4, consists of three parts, operating within separate process threads.

The CE evaluation thread concurrently evaluates each CE expression submitted to the system. A number of evaluations may also be active at the same time for a single CE expression. As each new event is received, it is determined whether it satisfies any part of an active evaluation, and whether it can initiate a new evaluation. When an evaluation accepts, the calling application is informed with the resulting event path, and internal garbage collection is performed.

Concurrent with event monitoring, two other process threads exist. One is responsible for producing timer events, the specification of which are submitted by the evaluation thread. The other thread manages buffering and re-ordering of incoming events. When the application supplied buffering delay has occurred, the events are propagated to the suitable evaluation control thread.



Figure 9.4: CE Monitoring Threads

9.9 Composite Event System Evaluation

This section will evaluate how the proposed composite event system can be used to build applications. A set of simple examples is firstly given, with a brief description of each and the way each is evaluated. In a similar way to chapter 8, an extended example is then described in detail. Finally, the performance of the CE system is briefly evaluated.

9.9.1 Composite Event Examples

As described previously an application may use the composite event system in two ways. Recall that the calling application submits the CE expression and a reference to a call-back object to be invoked when the expression is satisfied. An application may link at compile time with the CE library and then bind locally to the appropriate CORBA object. Alternatively an application may bind to a remote CE service and submit its CE expression. The former distributes computational load, the latter is of greater convenience to applications. The following examples use this first method.

1. Active Badge application control.

The functionality of the Active Badge as a control device can be extended by monitoring for a number of button presses within a certain time period. For example:

 This expression firstly monitors for any badge event from a badge associated with a person where the side button has been pressed. The initial => follow-by operator allows multiple evaluation threads to be active at the same time, therefore allowing multiple badges to be monitored. The second part of the expression then monitors for another badge event from the same badge. Evaluation accepts only if the second event occurs with 5 seconds of the first. If the timer expires the evaluation is rejected.

Such an expression may be used to launch an application or to indicate when a person does not wish to be disturbed.

2. User Activity

By monitoring workstations it is possible to infer whether a user is active. Consider the following CE expression:

This monitors for workstation activity caused by 'gjn' *and* badge events from 'gjn'. Badge events are used to help validate the correctness of the information. If 'gjn' is seen to be active at a workstation and is in the building, it is more likely that this reported activity will be correct, rather than the report referring to activity by some other user at a workstation where 'gjn' happens to be logged on.

a $^{\circ}$ b means that the CE will fire if both the events *a* and *b* occur, *in any* order. Note that there is no time constraint within which the two events must occur. This may prove problematic. For example, consider if the last badge sighting produced for 'gjn' was at 9am. If workstation activity is then reported at 3.30pm a CE will occur comprising these two events. An application may not wish to initiate an action based upon these temporally remote events. To address this problem either a time constraint must be introduced into the CE expression or the application must itself examine the time-stamps on the events comprising the composite event.

To do the former the CE expression would need re-writing as:

This expression will only accept if a workstation activity and badge event are seen within a time window of 20 seconds.

3. Security Monitoring

The ability to monitor the environment enables security applications to be implemented. Each member of staff at ORL wears an Active Badge. The absence of badge events gives a strong indication that no one is present in the building. Consider the following CE expression:

This expression initially monitors for badge events. After a period of 600 seconds (10 minutes) in which no badge events have occurred anywhere in the ORL building, video motion and door activity are monitored for. The CE accepts if either one of these events occurs without a badge event. The application may then, for example, inform a member of security personnel that unauthorised movement has been seen within the building.

4. A Context-Aware Telephone Directory

A commonly used application at ORL is one displaying Active Badge information which also shows a user's nearest telephone. This acts as a dynamic telephone directory. A convenient application is one which displays this telephone directory when a telephone handset is picked-up on the nearest display.

To facilitate this, consider the following CE expression:

This firstly looks for any telephone that has been picked-up. It then waits for a *proximity* event indicating that a piece of equipment is within 2mof the telephone. Finally, a state lookup is made to determine whether the equipment has been idle for at least 60 seconds. The CE will reject if, after the initiating phone 'pickup' event, the same phone's handset is put back down.

Note the use of the => follow-by operator, placed after the initial 'Phone' event. This enables the evaluation to deal with multiple proximity events, some of which may occur when unsuitable items of equipment are seen near the telephone, and where the resulting activity lookup fails.

This example requires *proximity* to be defined as an event type. This is described below in section 9.9.2.

5. Seminar Monitoring

The following example demonstrates how a seminar may be monitored. Consider a seminar room which is normally kept locked (possibly to protect valuable equipment or to prevent non-seminar use). The seminar organiser will typically arrive a few minutes before to prepare, leaving the door open to welcome attendees. The door will be closed as the speaker starts indicating that the seminar should not be disturbed. The following CE expression monitors this scenario:

A set of events will occur when the seminar room door is finally closed. By examination of the event paths the application can determine those seen within the seminar room, between the time at which the door was initially opened and the time it was closed. This therefore could be used by the seminar organiser to record who attended, or by a tea-person to organise sufficient provisions for the number of people.

9.9.2 An Extended Example

To illustrate how event monitoring and composition may be used within a real application, *promiscuous teleporting*, described earlier in chapter 8, will be considered again. A problem with the present system is that each user's teleporting agent only pays attention to nearby devices and whether they are capable of accepting a teleport session.

This is not sufficient. For example, consider if user A is near a teleport capable display. If this display is currently being used by user B it is inappropriate if user A's promiscuous teleporting session materialises.

Access to Proximity Events

The promiscuous teleporting example given in section 8.1.5 used the appropriate native CORBA interface of the location service. By modelling proximity as an event this interface may be accessed within a CE expression.

To do this a CALAIS event service is built using the proximity interface of the location service as the low-level event source. The event interface is effectively a wrapper around the location service. The proximity interface provides the following interface:

object type: the type of the object the proximity query is to be bound to;

- **object identifier:** the name of the object the proximity query is to be bound to;
- **distance:** the distance from the object the spatial query will extend to. The single distance is applied in each dimension;
- error: the maximum acceptable spatial error;
- **matching object type:** the type of the objects that will be matched by the query;

matching object identifier: the identifier of the object the query is to match.

Thus Proximity('PERSON', 'gjn', '2.0', error, type, id) will match on any object that is seen within 2m of person 'gjn' with any error.

State lookup functionality will be provided by a wrapper around a query interface to the location service rather than by using the default CALAIS event store mechanism.

Monitoring Teleport Sessions

To monitor teleport sessions, another new event type is introduced. This has the following interface:

Teleport(display_type, display_id, userid, teleport_id, action)

display type: the type of the display the session has teleported to;

display identifier: the identifier of the display the session has teleported to;

userid: the user-id of the person owning the teleport session;

teleport identifier: the teleport session identifier;

action: 'MATERIALISE' or 'DEMATERIALISE'.

Thus, whenever a teleport session materialises or de-materialises to, or from a display, an event is produced indicating the name of the relevant display and the person owning the teleport session.

Publishing New Event Services

Once a new event service has been created, its address can be entered into the CALAIS event type repository. At CE system initialisation (section 9.8.2) the new event type template can be read, and the event service is then available to applications.

A CE Expression

The following expression adds some intelligence to the promiscuous teleporting application:

This expression firstly monitors for proximity events, specifically those objects which are seen within 2m of user 'gin' with a location error of less than 1m. For each proximity event received, state lookups are used to ascertain whether the display is currently idle and no teleport session is present, or the teleport session was materialised more than 60 seconds ago. If either of these cases is true the composite event evaluation accepts, invoking the application's call-back object. The application can then teleport the appropriate session to the display.

9.9.3 CE System Performance

The CE library has been implemented on the DEC Alpha architecture, running Digital Unix. CE evaluation proves to be efficient. Within a single application process and on the receiving of a single event, approximately 400 separate event evaluations can be performed per second. This level of performance ensures that the real-time requirements demanded by many applications will be met.

9.9.4 Wider Application of Composite Events

So far, the examples given have shown the CE system being applied within a particular application domain. However the event paradigm and composite event evaluation have broad application in many domains. Two examples are:

1. Home Device Control

The television volume is required to be muted if the telephone or the door bell rings.

This expression starts monitoring when a television is switched on. It monitors independently for each television. The CE accepts when a telephone or door-bell rings without the television being switched off first.

2. Fire Alarm Evacuation

When a fire-alarm activates, it is helpful to monitor the people seen inside the building and leaving to determine whether anyone is in difficulty. The following expression is helpful:

```
fire -> FireAlarm('ACTIVATE') => PersonSeen(P) ->
    FireAlarm('DEACTIVATE') - PersonLeaves(P);
```

Monitoring starts when the fire-alarm is activated. Each time a person is seen within the building (using an Active Badge for example) a new monitoring thread is started. The CE accepts when the fire-alarm is deactivated (when the fire-brigade arrives for example) only if the person has not been seen leaving. When the fire-alarm is de-activated a set of call-backs will occur, one for each person seen within the building but *not* seen leaving.

9.10 Summary

A composite event system and associated language have been presented as a method of binding different event sources together and to allow applications to conveniently express their interest in events. A set of examples has demonstrated its effective use within the target application domain. In addition some scenarios from other potential application domains were described. 122

Chapter 10

Conclusions

10.1 Summary

This thesis has proposed CALAIS, an architecture suitable for the support of context-aware applications operating within a typical indoor, office domain.

Chapter 1 outlined the research area and identified the benefits to applications that context- aware monitoring of the environment can provide. Current approaches, and related research were reviewed in chapter 2. Related work involves location sensing systems, strategies for the management of the resulting data, and approaches to the sensing of the physical environment. Application areas were also identified, together with strategies for the delivery of data to applications. Chapter 3 described shortcomings in current approaches which then lead to the motivations for the research described in this thesis. The following requirements were identified:

- An abstracted view of information produced by sensors monitoring the environment. Current approaches utilise ad-hoc interfaces to sensor technology. These make applications difficult to construct and prevent extensibility.
- Generic model of location information. Location information will be produced by a range of location sensing devices. Current approaches are typically designed around a single technology.
- Location information management, enabling the asynchronous monitoring of spatial regions. Current application interfaces are rudimentary and typically do not allow spatial querying.
- Representation of the static physical environment to enhance locationbased querying capability.
- Integration of access to persistent resources, such as those representing the capabilities of mobile equipment and the layout of the physical domain.

• Application support enabling access to multiple sources of environmental monitoring information. This facilitates the rapid prototyping of context-aware applications.

These areas provide the focus of later chapters.

Chapter 4 presented a set of deployed and prototype sensor systems suitable for the monitoring of an office environment. The output from these formed the experimental data used in this thesis. Chapter 5 proposed a flexible abstraction of a sensor system which enables sensor information to be delivered asynchronously to applications. The CORBA distributed systems architecture was used to enable communication in a distributed environment and the abstraction of interfaces away from supporting hardware and software. Automated tools were described which enable the implementation of an event system from a declaration of its interface.

Chapter 6 identifies the facilities required in a location service. These were:

- 1. Integration of information from multiple location sensor sources.
- 2. Support for spatial queries executed both synchronously and asynchronously.
- 3. Support for representing the orientations of objects.
- 4. The representation of room boundaries and the connections between rooms.
- 5. High performance spatial indexing enabling the management of thousands of physical object locations.

Chapter 7 described the implementation of a location service with the required facilities. This resulted in a lightweight, high-performance system with powerful querying capabilities. An evaluation of this system was presented in chapter 8. This illustrated the use of a number of deployed location sensor systems and demonstrated, by example, the query facilities available, as well as showing that the implementation can handle the representation of thousands of objects and the execution of thousands of queries per second.

Attention was turned to application support in chapter 9. A simple, flexible and powerful language was proposed which enables applications to easily specify context-aware scenarios. The run-time system handles the delivery of events from multiple, distributed sensor systems, and deals appropriately with issues such as network delays. Again, a set of examples were described which demonstrated the worth of the approach.

10.2 Potential Further Work

The area of context-aware systems is a broad one and still largely undeveloped. This thesis has explored some of the technologies necessary to practically realise
such a system in a typical target domain. It is envisaged that some appropriate areas for future work are:

• Location technologies

A new ultrasound location technology [WJH97] has been used within the experimental part of this thesis. It is currently in prototype form, and experience so far suggests it may eventually replace, or at least augment, the deployed Active Badge system. Its effectiveness requires wider deployment and a substantial user base. This will allow a more realistic determination of its capabilities and, moreover, will expose previously unconsidered application areas.

• Event storage

CALAIS event services store a small amount of state, sufficient to support applications needing to know *most recent* events. The storage of greater numbers of past events may be necessary for some types of application, such as time and motion studies, stock-market analyses and dynamic resource allocation schemes which may attempt to predict an event based upon previous occurrences. Re-enactment of "a sequence of events" would also be possible.

• Persistent storage of location information

The data structures used to support the proposed location service are all held in main memory, which is volatile. Although this was done to ensure a high level of performance, much system state would be lost if the machine on which the service was running failed. Investigation into a lightweight, disk based system may therefore be appropriate. This will result in the issues of concurrency and transaction management needing to be addressed.

• Scalability

The evaluation of the location service presented in chapter 8 concludes that it is suitable for handling thousands of objects and hundreds of updates per second. A single location service, consisting of a single spatial index is considered for a single domain. However, very large domains, with high numbers of objects, may cause the performance limit which can be handled by a single location service to be exceeded. In these circumstances the use of multiple services would be required, perhaps arranged to present a single logical service. A hierarchical scheme is likely to be suitable, with a physical domain divided into sub-domains, each covered by a single service. A hand-off protocol would facilitate the movement of objects between sub-domains. Spatial queries spanning more than one sub-domain would inevitably be more costly to execute than those spanning a single domain.

• Inter-domain Working

The issue of inter-domain working has not been addressed in detail by

CALAIS. However, its importance as a requirement for a system supporting mobility has been recognised. Inter-domain issues such as naming and the distribution of mobile object attributes need to be addressed.

Bibliography

- [AL94] N Ackroyd and R Lorimer. Global Navigation: a GPS User's Guide. Lloyd's of London, 2nd edition, 1994.
- [APM90] Architecture Projects Management Limited, Poseidon House, Castle Park, Cambridge, UK. Advanced Networked Systems Architecture, August 1990.
- [ASH97] N Adly, P Steggles, and A Harter. SPIRIT: a Resource Database for Mobile Users. In Proceedings of ACM CHI 1997 Workshop on Ubiquitous Computing, Atlanta, Georgia, USA, 1997.
- [ATC97] Flock of Birds. Technical report, Ascension Technology Corporation, 1997. http://www.ascension-tech.com/products.htm/.
- [Azu93] R Azuma. Tracking Requirements for Augmented Reality. Communications of the ACM, 36(7):50-51, July 1993.
- [Bat94] J Bates. Presentation Support for Distributed Multimedia Applications. PhD thesis, University of Cambridge Computer Laboratory, Cambridge, UK, 1994.
- [BBH97] J Bacon, J Bates, and D Halls. Location-Oriented Multimedia. *IEEE Personal Communications*, pages 48–56, October 1997.
- [BBHM95] J Bacon, J Bates, R Hayton, and K Moody. Using Events to Build Distributed Applications. In Proceedings of the 2nd International Conference on Services for Distributed and Networked Environments, 1995.
- [BCK93] A Banerji, D L Cohn, and D C Kulkarni. Mobile Computing Personae. Technical Report 93-5, Department of Computer Science and Engineering, University of Notre Dame, USA, May 1993.
- [Ben75] J L Bentley. Multi-dimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, 18(9):509–517, September 1975.
- [BHR94] F Bennett, A Harter, and T Richardson. Teleporting Making Applications Mobile. Technical report, Olivetti and Oracle Research Limited, Cambridge, UK, 1994.

- [Bro96] P J Brown. The Stick-e Document: a Framework for Creating Context-Aware Applications. *Electronic Publishing*, 9(1), September 1996.
- [BS98] J Bates and M Spiteri. A Store for Active Information. Technical report, University of Cambridge Computer Laboratory, Cambridge, UK, 1998.
- [Can93] S J Cannan. SQL the Standard Handbook. McGraw-Hill, 1993.
- [CKAK94] S Chakravarthy, V Krishnaprasad, E Anwar, and S K Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In Proceedings of the 20th VLDB Conference, pages 606– 617, Santiago, Chile, 1994.
- [CM93] S Chakravarthy and D Mishra. Snoop: An Expressive Event Specification Language for Active Databases. Technical Report UF-CIS-TR-93-007, University of Florida, USA, 1993.
- [EHC⁺93] S Elrod, G Hall, R Costanza, M Dixon, and J Des Rivieres. Responsive Office Environments. Communications of the ACM, 36(7):85– 86, July 1993.
- [Fit93] G W Fitzmaurice. Situated Information Spaces and Spatially Aware Palmtop Computers. Communications of the ACM, 36(7), July 1993.
- [FMS93] S F Feiner, B MacIntyre, and D Seligmann. Knowledge-Based Augmented Reality. Communications of the ACM, 36(7), July 1993.
- [Fra91] J Fraleigh. Calculus with Analytic Geometry. Addison-Wesley, 1991.
- [GJS92] N H Gehani, H V Jagadish, and O Shmueli. Composite Event Specification in Active Databases: Model & Implementation. In Proceedings of the 18th VLDB Conference, Vancouver, British Columbia, Canada, 1992.
- [Gut84] A Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In Proceedings of 1984 ACM-SIGMOD Conference on the Management of Data, pages 47–57, Boston, Massachusetts, USA, June 1984.
- [Hay96] R Hayton. OASIS: An Open Architecture for Secure Interworking Services. PhD thesis, University of Cambridge Computer Laboratory, Cambridge, UK, 1996.
- [HH94] A Harter and A Hopper. A Distributed Location System for the Active Office. *IEEE Network*, January 1994.
- [HL94] S Hodges and G Louie. Towards the Interative Office. Human Factors in Computing Systems, CHI Conference Companion, April 1994.

- [IB92] T Imielinski and B R Badrinath. Querying in Highly Mobile Distributed Environments. In Proceedings of the 18th Conference on Very Large Data Bases, pages 41–52, August 1992.
- [IB94] T Imielinski and B R Badrinath. Wireless mobile computing : Challenges in data management. Communications of the ACM, pages 19–27, October 1994.
- [Ill95] Illustra Information Technologies Inc, Oakland, CA, USA. Illustra User's Guide, 1995.
- [Kop92] H Kopetz. Sparse Time Versus Dense Time in Distributed Real-time Systems. In Proceedings of the 12th International Conference on Distributed Computing Systems, pages 460–467, Yokohama, Japan, June 1992.
- [KVP94] P Krishna, N H Vaidya, and D K Pradhan. Static and Dynamic Location Management in Distributed Mobile Environments. In Proceedings of the Third International Conference on Parallel and Distributed Information Systems, Austin, Texas, USA, September 1994.
- [LBC⁺94] M Lamming, P Brown, K Carter, M Eldridge, M Flynn, G Louie, P Robinson, and A Sellen. The Design of a Human Memory Prosthesis. The Computer Journal, 37(3), 1994.
- [Leo95] U Leonhardt. An Integrated and Federated Location Service. In IEE Colloquium on Mobile Computing and its Applications, London, UK, November 1995. IEE.
- [Mil91] D L Mills. Internet Time Sychronization: the Network Time Protocol. IEEE Transactions on Communications, pages 1482–1492, October 1991.
- [Moc87] P Mockapetris. Domain Names Concepts and Facilities. RFC 1034, November 1987.
- [NEL91] W M Newman, M Eldridge, and M G Lamming. PEPYS: Generating Autobiographies by Automatic Tracking. In Proceedings of the 2nd European Conference on Computer-Supported Cooperative Work, 1991.
- [Nel96] G J Nelson. User Interaction with Computers on the Move. Computers in Industry, 4(6), April 1996.
- [Nye92] A Nye. X Window System User's Guide. O'Reilly and Associates Inc, 1992.
- [OMG95] The Common Object Request Broker: Architecture and Specification. Revision 2.0. Technical report, Object Management Group, 1995.

- [OMG96] Common Object Services Specification. Technical report, Object Management Group, 1996.
- [RLU94] M Rizzo, P F Linington, and I A Utting. Integration of Location Services in the Open Distributed Office. Technical Report 14-94, Computing Laboratory, University of Kent, UK, 1994.
- [Sam88] H Samet. Hierarchical Representations of Collections of Small Rectangles. ACM Computing Surveys, 20(4), December 1988.
- [Sch95] W N Schilit. A System Architecture for Context-Aware Mobile Computing. PhD thesis, Columbia University, USA, 1995.
- [Sch96] S Schwiderski. Monitoring the Behaviour of Distributed Systems. PhD thesis, University of Cambridge Computer Laboratory, Cambridge, UK, 1996.
- [SMR⁺97] T Starner, S Mann, B Rhodes, J Levine, et al. Augmented Reality Through Wearable Computing. Technical Report 397, MIT Media Laboratory, Cambridge, Massachusetts, USA, 1997.
- [SR90] Y C Shim and C V Ramamoorthy. Monitoring and Control of Distributed Systems. In First International Conference on Systems Integration, pages 672–681. IEEE Computing Press, 1990.
- [SRF87] T Sellis, N Roussopoulos, and C Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In Proceedings of the 13th VLDB Conference, Brighton, UK, 1987.
- [SRH90] M Stonebraker, L Rowe, and M Hirohama. The Implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*, 2(7):125–142, March 1990.
- [ST93] M Spreitzer and M Theimer. Providing Location Information in a Ubiquitous Computing Environment. In Proceedings of the 14th ACM Symposium on Operating Systems Principles, December 1993.
- [ST94] B N Schilit and M M Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, pages 22–32, September/October 1994.
- [The96] D G Theriault. Smallworld GIS: An Open System Architecture for a Workstation-based Geographical Information System. Technical Paper 3, Smallworld Systems Ltd, Cambridge, UK, 1996.
- [vOV91] P van Oosterom and T Vijlbrief. Building a GIS on Top of the Open DBMS POSTGRES. In Proceedings of EGIS '91, Brussels, Belgium, April 1991.
- [Wal96] R Walters. What is CTI? Technical report, CTINet, http://www.ctinet.co.uk/, July 1996.

- [War95] A Ward. ORL Active Map, 1995. http://www.orl.co.uk/cgibin/mapgen/.
- [WB97] G Welling and B R Badrinath. A Framework for Environment Aware Mobile Applications. In Proceedings of the 17th ICDCS Conference, Baltimore, Maryland, USA, 1997.
- [WC96] J Widom and S Ceri, editors. Active Database Systems: triggers and rules for advanced database processing. Morgan Kaufmann, 1996.
- [Wei93] M Weiser. Some Computer Science Issues in Ubiquitous Computing. Communications of the ACM, 36(7):65-84, July 1993.
- [WGH94] S Wray, T Glauert, and A Hopper. The Medusa Applications Environment. Technical Report 94-3, Olivetti and Oracle Research Limited, Cambridge, UK, 1994.
- [WH92] R Want and A Hopper. Active Badges and Personal Interactive Computing Objects. Technical Report 92-2, Olivetti and Oracle Research Limited, Cambridge, UK, 1992.
- [WHFG92] R Want, A Hopper, V Falcao, and J Gibbons. The Active Badge Location System. Technical Report 92-1, Olivetti and Oracle Research Limited, Cambridge, UK, 1992.
- [WJH97] A Ward, A Jones, and A Hopper. A New Location Technique for the Active Office. *IEEE Personal Communications*, pages 42–47, October 1997.
- [Wor95] M F Worboys. GIS: A Computing Perspective. Taylor and Francis, 1995.
- [WRB⁺97] K R Wood, T Richardson, F Bennett, A Harter, and A Hopper. Global Teleporting with Java: Toward Ubiquitous Personalized Computing. *IEEE Computer*, pages 53–59, February 1997.

Appendix A

Event Source IDL

A.1 Generic Interfaces (register.idl)

A.2 Active Badge

```
#include "register.idl"
typedef struct badge_data_object_ {
    long sighting_type;
    string id;
    string id_domain;
    string room_id;
    string room_domain;
    long click;
} badge_data_object;
typedef sequence<badge_data_object> badge_data_object_seq;
interface badge_callback_object {
```

A.3 Ultrasound Badge

```
#include "register.idl"
typedef struct ultrasound_data_object_ {
  string id;
 float x, y, z;
  string coordsys;
} ultrasound_data_object;
interface ultrasound_callback_object {
 oneway void notify(in ultrasound_data_object data, in long secs,
                     in long usecs);
};
interface ultrasound_server_object {
 long register_callback(in ultrasound_callback_object callback_obj,
                         in ultrasound_data_object signature);
  void report_state(in ultrasound_data_object signature,
                    out ultrasound_data_object_seq state
                    in long secs, in long usecs);
 long deregister(in long client_num);
};
```

A.4 Workstation Activity Monitor

```
#include "register.idl"
typedef struct xmonitor_data_object_ {
   string id;
   string host_domain;
   string userid;
```

A.5 Door Monitoring

```
#include "register.idl"
typedef struct door_data_object_ {
        string id;
        string domain;
        long value;
        long secs;
        long usecs;
} door_data_object;
typedef sequence<door_data_object> door_data_object_seq;
interface door_callback_object {
 oneway void notify(in door_data_object data, in long secs,
                     in long usecs);
};
interface door_server_object {
 long register_callback(in door_callback_object callback_obj,
                         in door_data_object signature);
 void report_state(in door_data_object signature,
                    out door_data_object_seq state
                    in long secs, in long usecs);
 long deregister(in long client_num);
};
```

A.6 Telephone Handset Monitoring

```
#include "register.idl"
typedef struct phone_data_object_ {
  string id;
  string domain;
  long status;
} phone_data_object;
typedef sequence<phone_data_object> phone_data_object_seq;
interface phone_callback_object {
 oneway void notify(in phone_data_object data, in long secs,
                     in long usecs);
};
interface phone_server_object {
 long register_callback(in phone_callback_object callback_obj,
                         in phone_data_object signature);
  void report_state(in phone_data_object signature,
                    out phone_data_object_seq state
                    in long secs, in long usecs);
 long deregister(in long client_num);
};
```

A.7 Motion Detection

```
#include "register.idl"
typedef struct motion_data_object_ {
  string id;
  string domain;
} motion_data_object;
typedef sequence<motion_data_object> motion_data_object_seq;
interface motion_callback_object {
 oneway void notify(in motion_data_object data, in long secs,
                     in long usecs);
};
interface motion_server_object {
 long register_callback(in motion_callback_object callback_obj,
                         in motion_data_object signature);
  void report_state(in motion_data_object signature,
                    out motion_data_object_seq state
                    in long secs, in long usecs);
 long deregister(in long client_num);
};
```

136

Appendix B

Container and Junction Database Entries

B.1 ORL Floor 1 Containers

container name; min x,y,z max x,y,z coordinates of container in relation to domain origin

R101 21.090909 23.909091 0.0 29.727273 27.818182 3.0 R103 18.090909 23.636364 0.0 21.090909 26.909091 3.0 R104 15.363636 23.636364 0.0 18.090909 26.909091 3.0 R105 12.454545 23.636364 0.0 15.363636 26.909091 3.0 R106 9.636364 23.636364 0.0 12.454545 26.909091 3.0 R107a 7.000000 23.636364 0.0 9.636364 26.909091 3.0 R107;1 7.000000 18.818182 0.0 9.636364 23.636364 3.0 R107;2 1.000000 18.818182 0.0 7.000000 26.909091 3.0 R108 9.636364 18.818182 0.0 12.454545 22.000000 3.0 R109 12.454545 18.818182 0.0 15.363636 22.000000 3.0 R110 15.363636 18.818182 0.0 18.545455 22.000000 3.0 R111 18.545455 18.818182 0.0 21.090909 22.000000 3.0 R112 22.818182 16.818182 0.0 25.272727 19.727273 3.0 R113 25.272727 16.818182 0.0 28.727273 21.727273 3.0 R114 22.818182 19.727273 0.0 25.272727 21.727273 3.0 F1HALL;1 9.636364 22.000000 0.0 21.090909 23.636364 3.0 F1HALL; 2 21.090909 21.727273 0.0 29.727273 23.909091 3.0 F1HALL; 3 21.090909 16.818182 0.0 22.818182 21.727273 3.0

B.2 ORL Floor 1 Junctions

R101 24.545455 23.909091 0.0 25.545455 23.909091 3.0 R103 19.454545 23.636364 0.0 20.181818 23.636364 3.0 R104 16.363636 23.636364 0.0 18.090900 23.636364 3.0 R105 12.818182 23.636364 0.0 14.454545 23.636364 3.0 R106 9.636365 23.636364 0.0 11.272727 23.636364 3.0 R107a 7.909091 23.636364 0.0 8.954545 23.636364 3.0 R1079.63636422.1818180.09.63636423.4545453.0R1089.63636522.0000000.011.27272722.0000003.0R10912.72727322.0000000.014.45454522.0000003.0R11016.27272722.0000000.017.09090922.0000003.0R11119.36363622.0000000.020.09090922.0000003.0R11222.81818218.5454550.022.81818219.5454553.0R11325.72727321.7272730.026.63636421.7272733.0R11424.09090921.7272730.024.81818221.7272733.0

B.3 ORL Floor 1 IR-Zones

name of originating badge place, coordinates of IR zone

R101 21.090909 23.909091 0.0 29.727273 27.818182 3.0 R103 18.090909 23.636364 0.0 21.090909 26.909091 3.0 R104 15.363636 23.636364 0.0 18.090909 26.909091 3.0 R105 12.454545 23.636364 0.0 15.363636 26.909091 3.0 R106 9.636364 23.636364 0.0 12.454545 26.909091 3.0 R107a 7.000000 23.636364 0.0 9.636364 26.909091 3.0 R107;1 7.000000 18.818182 0.0 9.636364 23.636364 3.0 R107;2 1.000000 18.818182 0.0 7.000000 26.909091 3.0 R108 9.636364 18.818182 0.0 12.454545 22.000000 3.0 R109 12.454545 18.818182 0.0 15.363636 22.000000 3.0 R110 15.363636 18.818182 0.0 18.545455 22.000000 3.0 R111 18.545455 18.818182 0.0 21.090909 22.000000 3.0 R112 22.818182 16.818182 0.0 25.272727 19.727273 3.0 R113 25.272727 16.818182 0.0 28.727273 21.727273 3.0 R114 22.818182 19.727273 0.0 25.272727 21.727273 3.0 F1HALL;1 9.636364 22.000000 0.0 21.090909 23.636364 3.0 F1HALL; 2 21.090909 21.727273 0.0 29.727273 23.909091 3.0 F1HALL;3 21.090909 16.818182 0.0 22.818182 21.727273 3.0