Visual Sensing and Middleware Support for Sentient Computing

Diego López de Ipiña González de Artaza

Downing College
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

January 2002

To my beloved fiancée Ingeborg for all her patience and support.

A mis padres Fermín y Araceli por su continuo apoyo y dedicación.

Eusko Jaurlaritzari bere diru laguntzeagatik.

Abstract

Offices and homes are increasingly becoming populated with computing devices that are usually networked in some form. This ubiquitous deployment of computing technology is being made possible through the advances experienced in hardware and networking and the more attainable prices. Although computing devices have become more sophisticated and powerful, the way humans use them remains the same, *i.e.* a desktop-bound model. This gap between the user and the machine has remained unaltered since personal computers were first introduced two decades ago. This research work aims to promote the deployment of a new paradigm of computing, namely *Sentient Computing*, intended to reduce this gap.

Sentient Computing provides computing systems with awareness of their surrounding environment so that they can change their behaviour according to the observed situation. Rather than forcing the user to explicitly input information, the devices themselves perceive the dynamic world they are part of through sensors. Thus, it is possible to build systems that are attentive to user activities by obtaining even elementary notions of *context*, *e.g.* location, identity, proximity or activity. Computer systems and applications, when provided with a certain degree of perception, are made more natural, flexible, and adaptable.

Among the types of context information provided by sensors, *location* has proven to be especially useful. Since location is an important context that changes whenever the user moves, a reliable location-tracking system is critical to many context-aware applications. However, the sensor technologies conventionally used in indoor location tracking are expensive and complex to deploy, configure and maintain. These factors have prevented a wider adoption of Sentient Computing in our living and working spaces. This dissertation presents a low-cost and easily deployable vision-based sensor technology, which employs off-the-shelf hardware (web-cams and PCs) and printable circular markers for the identification and accurate 3D coordinate location and orientation of tagged entities. Users with a web-cam attached to their PCs simply need to install the software of this sensor in order to provide visual awareness to their systems.

Subsequently, a set of programming abstractions for Sentient Computing are proposed that enable the developer to simply concentrate on the specification of the context needs and the implementation of the reactive behaviour of a particular sentient system. The details dealing with access to the raw data from sensors and its interpretation into the high-level context needs of applications are made transparent to the developer by the set of abstractions defined. Moreover, common functionality within sentient applications, such as the correlation of event-based context notifications and the activation, migration and deactivation of services as a consequence of sensed situations, is supported by network accessible middleware services. The thesis of this dissertation is that the set of technologies suggested for sentient systems, both in the sensing and infrastructure support areas, will enable their easier development and, more importantly, their later deployment even in spaces where before this was infeasible, such as our homes or workplaces.

Keywords: *context-aware computing, location sensor, image processing, computer vision, middleware, CORBA, event correlation, mobility, service lifecycle control, sentient spaces*.

Preface

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of work done in collaboration.

This dissertation is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other university.

This dissertation does not exceed 65,000 words, including appendixes, bibliography, footnotes, tables and equations. It does not contain more than 150 figures.

Diego López de Ipiña

January 28, 2002

Acknowledgements

I would like to thank my supervisor, Andy Hopper, for his guidance and encouragement during my research. Thanks are also due to Sai-Lai Lo, formerly at AT&T Laboratories Cambridge, whose advice during this research was invaluable.

I am indebted to the numerous colleagues and friends who provided inspiration and assistance during the course of this work. Especially Robert Headon, Hani Naguib, Ripduman Sohan and George Coulouris, at the Laboratory for Communications Engineering, for their willingness to participate in challenging discussions; Paulo Mendonça, from the Fallside Laboratory, for answering my innumerable questions about computer vision; and Joe Newman, at AT&T Laboratories Cambridge, for his help and encouragement during the initial stages of this research.

I am grateful to Robert Headon, Duncan Grisby, Rupert Curwen, Jamie Walch, Frank Stajano, Steve Hodges, Gray Girling, Quentin Stafford-Fraser and David Riddoch, who reviewed earlier drafts of this dissertation and made many useful comments. Thanks are also due to all my colleges in the Laboratory for Communications Engineering (LCE) for making my stay in Cambridge very enjoyable.

My PhD studies were generously sponsored by the Department of Education of the Basque Government. I am also very grateful to AT&T Laboratories Cambridge for the industrial sponsorship of this work, and to the Department of Engineering and Downing College for partially funding my attendance to some conferences. Finally, I want to thank my fiancée Ingeborg Dorsman for her continuous support (wonderful cooking!) and patience during the three years and four months this research work has lasted.

Publications

Aspects of the work described in this dissertation feature in the following publications:

Diego López de Ipiña, "Building Components for a Distributed Sentient Framework with Python and CORBA", *Proceedings of the 8th International Python Conference*, Arlington, VA, USA. January 24 - 27, 2000.

Diego López de Ipiña and Sai-Lai Lo, "LocALE: a Location-Aware Lifecycle Environment for Ubiquitous Computing", *Proceedings of the 15th International Conference on Information Networking (ICOIN-15)*, Beppu City, Japan. January 31 - February 2, 2001[1].

Diego López de Ipiña, "Video-Based Sensing for Wide Deployment of Sentient Spaces", *Proceedings of 2nd PACT 2001 Workshop on Ubiquitous Computing and Communications*, Barcelona, Catalonia, Spain, September 8-12, 2001.

Diego López de Ipiña and Sai-Lai Lo, "Sentient Computing for Everyone", *Proceedings of the 3rd International Conference on Distributed Applications and Interoperable Systems (DAIS'2001)*, Krakow, Poland, September 17-19, 2001.

Diego López de Ipiña and Eleftheria Katsiri, "An ECA Rule-Matching Service for Simpler Development of Reactive Applications", *Published as a supplement to the Proceedings of Middleware 2001 at IEEE Distributed Systems Online*, vol. 2, no. 7, November 2001.

Diego López de Ipiña, Paulo Mendonça and Andy Hopper, "TRIP: a Low-Cost Vision-Based Location System for Ubiquitous Computing", to appear in *Personal and Ubiquitous Computing* journal, Springer.

[1] This work was awarded with ICOIN-15's best student paper prize.

Contents

List of Figures

List of Tables

Glossary

2D	Two Dimensional
3D	Three Dimensional
3G	Third Generation Mobile Telecommunications Systems
AI	Artificial Intelligence
AMS	Active Map Server
AOA	Angle Of Arrival
AR	Augmented Reality
CA	Context Abstractor
CC	Context Channel
CCD	Charge-Coupled Device
CCTV	Closed-Circuit Television
CG	Context Generator
CLIPS	C Language Integrated Production System
CORBA	Common Object Request Broker
CPU	Central Processing Unit
DC	Direct Current
DGPS	Differential Global Positioning System
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface
DTD	Document Type Definition
ECA	Event Condition Action
EGNOS	European Geo-stationary Navigation Service
ETR	Event Type Repository
FPS	Frames Per Second
GEM	Generalised Event Monitoring Language
GIOP	General Inter-ORB Protocol
GPS	Global Positioning System
GRF	Ground Reaction Force
HTTP	HyperText Transfer Protocol
ID	Identifier
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IMR	Implementation Repository

IOR	Interoperable Object Reference
IP	Internet Protocol
JESS	Java Expert System Shell
L3RF	Low range Long life Low cost Radio Frequency
LAN	Local Area Network
LCManager	Life Cycle Manager
LCServer	Life Cycle Server
LHS	Left Hand Side
LocALE	Location-Aware Lifecycle Environment
NTP	Network Time Protocol
OCR	Optical Character Recognition
OMG	Object Management Group
ORB	Object Request Broker
PDA	Personal Digital Assistant
POA	Portable Object Adapter
PTZ	Pan Tilt Zoom
RF	Radio Frequency
RFID	Radio Frequency Identification
RGB	Red-Green-Blue (Colour Model)
RHS	Right Hand Side
RPC	Remote Procedure Call
SA	Selective Availability
SGML	Standard Generalised Mark-up Language
SIF	Sentient Information Framework
SNR	Signal to Noise Ratio
TCP	Transport Control Protocol
TDOA	Time Difference Of Arrival
TDS	TRIP Directory Service
TI*RFID	Texas Instruments Radio Frequency Identification Systems
TRIP	Target Recognition using Image Processing
UMTS	Universal Mobile Telecommunications Systems
URL	Uniform Resource Locator
VNC	Virtual Network Computing
WAAS	Wide Area Augmentation System
XML	Extensible Mark-up Language

Chapter 1

Introduction

Progress in hardware miniaturisation, wireless networking, and sensor technologies is enabling computers to be used in more places and to have a greater awareness of the dynamic world they are part of. Over the last few years, a new class of *sentient* or *context-aware* systems has been developed, showing how computers can leverage even elementary notions of *context* such as location, identity or activity. These systems aim to reduce the *gap* between the physical world in which humans live, and the virtual world in which computers operate. The use of context information is especially important in a mobile environment, where the context, and thus the user's needs, changes frequently. Advances in both the technologies we use and the way in which we interact with them are driving us towards the *mobile and ubiquitous computing* paradigm, where users expect to access whatever information and services they want, whenever they want and wherever they are.

Humans use context to adapt their behaviour to the current circumstances. For example, if the ambient sound level is high we raise our voices to make ourselves heard. In contrast, computers are not as good as humans in deducing situational information from their environment and using it in interactions. Computers are better than humans at processing and outputting data, but they are much worse at capturing implicit input such as the mood of the user or their current location, and even worse at interpreting these inputs. Computer understanding has been the subject of research in Artificial Intelligence (AI) for the last 40 years, however humankind is still far away from being able to make computers 'think' [MIT01]. Through context, however, richer input to computers can be provided and, hence, a more flexible and natural way of interacting with them produced [Dey00].

The main aim of this dissertation is to promote the use of context information by computers in order to reduce the gap that separates humans from them. Unfortunately, context can be a rather generic term that may lead to contradictory interpretations. The next section provides a definition of what is meant by the term *context* within this dissertation.

1.1 Definition of Context

The term *context* is defined in the Cambridge International Dictionary of English as *"the influences and events that helped cause a particular event or situation to happen"*.

Several authors have tried to provide a formal definition of this term when applied to computing. In [Dey+00], a thorough analysis of other researchers' attempts to define context is given. As a result of this, Dey *et al*. suggest a good definition of context that is adopted by this work:

"Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves".

Some of the attributes that can be associated with the context of an entity are:

- Identity
- Spatial information (location, orientation, speed and acceleration)
- Temporal details (time of day, date and season of year)
- Environmental situation (temperature, air quality, light and noise level)
- Social interaction (who we are with and people that are nearby)
- Resources that are nearby (accessible devices and hosts)
- Capabilities of resources (display size, sound capabilities, video input)
- Physiological measurements (blood pressure, heart rate, respiration)
- Activity (talking, reading, walking, running)
- Schedules and agendas.

In fact, the combination of several context attributes may generate a more powerful picture of the current state of the world. By fusing several sensor readings, higher abstraction of context can be generated. Notably, some attributes of context such as location, identity, activity and time are more important than others and can act as indices into other sources of contextual information, as noted by [Dey+00]. For example, knowing the current location and current time, together with the user's calendar, the application will have a good idea of the user's current social situation, such as having a meeting, sitting in the class, waiting in the airport, and so on.

An application's operation is influenced in different ways depending on the characteristics of the incoming context. Two main categories of context can be defined: *active context* and *passive context*. Active context determines when and how the application or system should change its behaviour. Passive context influences and is relevant to the sentient system, but is not critical. Thus, there are essentially two ways in which context can be used by applications: (1) automatically adapt their behaviour according to discovered context (using active context) or (2) present the context to the user on the fly and/or store the context for the user to retrieve later (using passive context). This dissertation mainly exploits the first way of using context. Other authors [Dey00] have suggested a similar taxonomy of the ways in which context can be used by applications, distinguishing three main areas: (1) *presentation* of information and services to a user; (2) *automatic execution* of a service; and (3) *tagging of context* to information for later retrieval.

1.2 Sentient Computing

The use of context as a richer input to computers began to be investigated with the emergence of mobile computing components in the early 90s, led by the desire to support computer usage adequately in varying physical conditions. Early investigations were carried out at the Olivetti Research Lab[2] [Want+92] with the development of the Active Badge location system to adapt applications to people's whereabouts, and at Xerox

[2] Now AT&T Laboratories Cambridge.

PARC with the Ubiquitous Computing experiment, from which a first general consideration of context-aware [Schilit+94a] mobile computing emerged.

Weiser, in his seminal article on *Ubiquitous Computing* [Weiser92], set the basis of a new era of computing in which users' experiences with computers would become more calm and natural. He envisioned physical spaces, such as offices or homes, augmented with invisible computing devices integrated into the environment. The aim of Ubiquitous Computing is to make services provided by these embedded devices as commonly available and transparent to end-users as electricity. Ubiquitous Computing corresponds to the next generation of computing in which the user's attention will not be drawn to computers but vice versa. Computers will be attentive to user interactions and automatically aid their daily activities. Since Weiser's visionary thoughts, a lot of research effort has been spent on trying to make his Ubiquitous Computing dream a reality and so liberate the user from direct interaction, through mouse and keyboard, with 'that box on the table'. This dissertation presents an approach to address Ubiquitous Computing, namely Sentient Computing.

Sentient Computing [Hopper00][Addlesee+01] concerns the ability of computing systems to *detect*, *interpret* and *respond* to aspects of the user's local environment. Its goal is to enhance computer systems with a sense of the real world and give them as much information as the user about the aspects of the environment relevant to their application. To achieve this goal, it employs sensors distributed throughout the environment to maintain a detailed model of the real world and make it available to applications. Applications can then respond to environmental changes and automatically change their functionality, without explicit user intervention.

Sentient Computing is similar to other terms, such as *context-aware computing*, *pervasive computing*, or *smart spaces* found in the literature. Nevertheless, this work adopts the term *sentient* because it wants to stress the use of sensors as a means to add *perception*, *i.e.* capability to see or hear, to computer systems. Notably, the term *sentient* does not imply a need for the system to be intelligent or capable of forming new concepts about the world. The only goal of Sentient Computing is to give users the impression that computers have perception.

Research on Sentient Computing is driven by the emergence of low-cost and thus potentially widely available technologies that can provide inputs about the environment. Cameras, microphones and location systems such as the Global Positioning System (GPS) are possible sources of sentient data. By networking large numbers of such physical sensors and, in addition, acquiring through telemetry software information regarding the current state of computers, storage devices and networks, it is possible to conceive applications, devices and physical spaces that are highly reactive to the changing state of the physical world. Context-aware applications and devices might personalise themselves to the current user, alter their functionality based on where they were being used, or take advantage of nearby computing and communications resources.

1.3 The Importance of Location

Existing surveys [Abowd+01][Beadle+97][Korkea-aho00] on research in context-aware computing have shown that few attributes of context other than *location* are used in actual applications. In practice, few sentient applications have been developed that use attributes of context other than location and perhaps identity, activity and time. This strong focus on location has led to the definition of an interesting subset of Sentient Computing, named *Location-Aware Computing* [Want+01][Ward98]. A system is considered location-aware if its behaviour is dependant on the position of objects in the environment.

In order to enable Location-Aware Computing, the provision of a reliable location-tracking system is critical. In the outdoor environment, GPS currently provides 15 metre average location accuracy. In the indoor domain, several tracking technologies have been operating for several years providing variable location granularity. Unfortunately, whereas GPS has become very popular for developing outdoor context-aware systems, given its ubiquity and the lowering cost of GPS receivers, existing indoor location technologies present a variety of drawbacks which have prevented their widespread adoption in our living and working spaces. In particular, their high cost, custom-built hardware nature, and difficulties in their deployment and operation have prohibited a more widespread adoption.

This dissertation asserts the importance of location as a *sine qua non* attribute of context. This is the reason why a novel location technology is proposed that addresses the limitations of existing indoor location technologies. The aim is to demonstrate that usability levels as obtained with previous indoor location systems can be achieved by using existing off-the-self technology in an easier and more cost-effective way. However, it is disadvantageous to be constrained to the use of location information as a unique source of user context. Other sources of context [Schmidt+99], such as the current time or the activity somebody is engaged in, are of interest in order to permit computing systems to be attentive and so adapt their behaviour to the user. Therefore, this work also takes into consideration other kinds of context apart from location.

1.4 Infrastructure Support for Sentient Computing

The usage of context information in applications is still quite limited since it is very challenging and complex to capture, represent, and process. Contextual data is captured from heterogeneous and distributed sensors. Common context semantics must be agreed between applications and the raw data captured must be transformed into the forms understood by applications. Applications must embed some intelligence to process the information and to deduce meaning.

Past research on context-aware computing has usually failed to supply suitable modelling abstractions that will facilitate sentient applications design. Likewise, little has been done to provide infrastructure services that aid the programmer in the development and deployment of sentient applications. Most proposed solutions are intimately associated with the underlying sensing technology and do not adequately separate the context capture from context use. Developers have been forced to invest a lot of effort in the duplication of details that are common to most sentient applications. Only recently have other researchers considered these issues and have suggested toolkits [Dey00] and infrastructural support [Hong+01] to alleviate this problem.

This work aims to provide the software infrastructure that enables the developer simply to concentrate on the specification of the context needs and the implementation of the reactive behaviour desired for a particular sentient system. The details of access to the

raw data from sensors and its interpretation into the application high-level context needs are made transparent to the user by the set of programming abstractions proposed. Moreover, common functionality within sentient applications, such as the correlation of event-based context notifications and the activation, migration and deactivation of services as a consequence of sensed situations, is supported by network-based middleware services.

1.5 Research Statement

This dissertation proposes a novel vision-based sensor that uses off-the-shelf technology – conventional video cameras and PCs – with the aim of obtaining the identity and accurate three-dimensional (3D) location and orientation of printed circular markers in the field of view. This tagged vision-based indoor location system aims to provide a better trade-off between the scalability, infrastructure complexity, price and location accuracy than other predecessor technologies.

Furthermore, this work attempts to complement the low-cost and easy deployment of the location sensor defined with a set of programming abstractions and supporting middleware services that facilitate the design, development and deployment of sentient systems. Several applications using the visual sensing and middleware support contributions devised are illustrated in order to validate the following thesis statement:

"For a more widespread adoption of Sentient Computing in our homes or workplaces, it is necessary to define more easily-deployable location sensing technologies and middleware support addressing common functionality aspects of sentient systems such as contextual event correlation and service lifecycle and location control"

1.6 Research Scope

This research is oriented towards making *sentient environments* that are richly equipped with networked computing and sensing devices. Therefore, the focus of this work is the domain of *smart spaces* [Addlesee+01][Brumitt+00], where their inhabitants benefit from computerised services automatically triggered by a reactive environment

[Cooperstock+97]. This differs from the approach of other researchers [Kortuem+98] [Weatherall02] who couple mobile devices with the intelligent environment's infrastructure and use those devices as ubiquitous remote controllers for all the augmented devices.

This dissertation assumes augmented environments that somehow 'understand' the current context and as a consequence trigger adequate services aiding user activities. However, endowing a system with *cognition* or understanding is difficult, if not impossible, and is the challenge of Artificial Intelligence (AI). The approach followed to tackle this issue has been to adopt a rule-based programming paradigm. Simple rules can be applied to the rich contextual inputs in order to simulate a degree of intelligence within the system and so provide richer outputs from computers. Therefore, the only 'intelligence' sentient systems have to offer is the capability to undertake some rule-based interpretation of certain values of context. In essence, a sentient system does not need to be intelligent or capable of forming new concepts about the world – it only needs to respond to its perceptions in a way that is useful to the user.

1.7 Dissertation Outline

Chapter 2 overviews the state-of-the-art in Sentient Computing research. Sensing technologies, architectures to manage contextual data, and applications illustrating the potential of Sentient Computing are reported. This survey identifies the as-yet unresolved research issues that have prevented a more common adoption of the Sentient Computing paradigm.

Chapter 3 describes the design principles of a vision-based sensor technology and the image processing method employed for the successful identification of its markers in an image. An evaluation of the performance and accuracy of this method is also given.

Chapter 4 focuses on the 3D geometry required for the extraction of the position of objects marked by the circular barcodes used. After an overview of some essential projective geometry concepts, a detailed description of a method capable of extracting pose (3D location and orientation) from the image of a marker is given. Then, the

adaptive behaviour provided by this sensing device in order to provide real-time performance is described. The chapter concludes with a thorough examination of the accuracy and performance of the proposed technology.

Chapter 5 provides an insight into the event-based distributed architecture devised around the location technology proposed in order to export its sensorial data to interested parties. A careful examination of the video frame flow control issues triggered by the suggested image parsing process is also given. The work on making the vision sensor distributed has led to the development of a set of generic application building abstractions independent of the sensing technology used. A thorough description of these abstractions and their benefits on application development is given. This chapter also motivates the need for middleware services that enable the management of the set of contextual event correlations that define the logic of a sentient application, and of user-bound services' lifecycles and locations in a network.

Chapter 6 describes the design and implementation of a rule matching service for user specified Event-Condition-Action (ECA) rule management.

Chapter 7 elaborates on the middleware solution to user-bound service's lifecycle and location control proposed.

Chapter 8 validates all the contributions of this dissertation with the description of some applications that make use of both the sensor technology and the architectural support in the form of programming abstractions and middleware services that are defined.

Finally, chapter 9 presents a summary of the main contributions of this research and suggests potential future work.

Chapter 2

Related Work

This chapter reviews relevant research into Sentient Computing, with a special emphasis on Location-Aware Computing. Firstly, several sensor technologies used to determine the location of entities are examined. Since *location* is integral to many context-aware applications, the provision of a reliable location-tracking system is of paramount importance. Secondly, software architectures designed to efficiently manage both location and other types of context are reviewed. Suitable sensing technologies must be coupled with appropriate software support to transform and to disseminate the information captured to applications. Thirdly, illustrative applications showing the potential of this research area are discussed. Throughout the discussion of related work, issues not addressed by previous research that have prevented a more widespread adoption of Sentient Computing, are identified. This chapter concludes considering the social implications that may hinder the widespread adoption of Sentient Computing.

2.1 Location Technologies Overview

Section 1.3 noted the importance of the *location* attribute of context for the development of context-aware systems. When we move, our context changes; the available computing resources surrounding us vary, as do our social interactions (*e.g.* people that are nearby). This justifies the comprehensive research in systems that automatically locate people, equipment and other assets. Each proposed solution solves a different problem or supports different applications. They vary in many ways, such as the physical phenomena used for location determination, the form factor of the sensing apparatus, power requirements, infrastructural requirements, and resolution in time and space, *i.e.* how often an object's location is tracked and with what accuracy objects can be located.

Ward [Ward98] states that a good location technology should provide fine-grained spatial information at a high update rate, be unobtrusive (small, lightweight and wireless), scalable (to allow the location of many objects in a wide area), low-power, software-supported and low-cost. An interesting range of technologies has been devised in the last ten years satisfying these requirements to various extents. The following sections give an overview of existing location sensors, classified by the physical phenomena used for location determination. The following criteria, suggested by [Hightower+01] in their definition of a taxonomy for location systems, is used when describing those technologies:

- *Physical phenomena* or technology used to extract location, *e.g.* infrared, ultrasound, radio, and so on.
- *Tagged* or *untagged* nature of the objects that should be located; if tagged, whether the tags require battery power or not.
- *Identification/Orientation, i.e.* does the location system provide not only the position of objects but also their identity and orientation?
- *Physical* or *symbolic* location, *i.e.* does the location system provide the location information in terms of a frame of reference or as an abstract idea of where something is, *e.g.* next to Diego's computer?
- *Localised* vs. *centralised* location computation, *i.e.* is the location information calculated by a centralised infrastructure or is the mobile positioning device itself extracting the information?
- *Accuracy*/*Precision*, *i.e.* how accurate is the location information returned, *e.g.* within 3 cm, and with what frequency is that result returned, *e.g.* 95% of cases?
- *Cost* of the infrastructure required to deploy the system and its use of either *off-the-shelf* or *bespoke* technology.
- *Limitations* of the location system.

2.1.1 Infrared-based Technologies

2.1.1.1 Active Badge

The Active Badge System [Want+92], developed at the Olivetti Research Laboratory[3], pioneered research into indoor location systems. An Active Badge, depicted in Figure 2.1, is a wearable device 6x6x1 cm in size that transmits a globally unique code approximately every 10 seconds using a diffuse infrared data link. A simpler version of the Active Badge is also designed for tagging equipment. Each walled space within a building is equipped with one or more networked infrared sensors that detect badge transmissions. The location of the badge wearer is determined on the basis of the spatial region where the detecting sensors are contained. Since infrared signals do not travel through walls, this technology offers room resolution location. The personnel badges contain two push-buttons to permit simple user input. The resolution of this location system can be improved from room-scale to desk-scale granularity when a hybrid radio/infrared scheme is used as proposed by [Harter+94].

Figure 2.1. Active Badge tag.

2.1.1.2 PARCTab

The PARCTab [Want+95] experiment is widely cited as the first realisation in the field of Ubiquitous Computing. The goal of this project was to provide users with the ability to

[3] Now AT&T Laboratories Cambridge.

access computing resources in an un-tethered mobile way. A PARCTab is a handheld dumb terminal with a 128x64 pixel touch sensitive display, three buttons and a speaker. It uses an infrared-based cellular network for communication. In order to use a PARCTab within a building, each room must be equipped with an infrared transceiver, similar to those of the Active Badge system, which handles communication with all PARCTabs in the room. PARCTabs act as thin display clients for applications running on fixed machines on the LAN and as active badges emitting infrared signals picked up by room transceivers and used to provide location with room-scale granularity.

2.1.1.3 Smart Badge

A Smart Badge [Beadle+98] is a wearable device like an Active Badge, but with a collection of sensors and actuators and re-programmability added. The attached sensors measure environmental factors such as temperature, humidity, ambient light level, orientation and sound. The data collected from the sensors together with the unique identifier assigned to each Smart Badge are broadcast periodically across an infrared interface to networked sensors placed around a building. Smart Badges also have an output port to which computing devices can be attached, and to which data can be sent from the fixed sensor network. The main motivation of this active tag is to extend the application scope of the Active Badge, from location-aware to the broader context-aware computing area.

2.1.1.4 Locust Swarm

The Locust Swarm [Kirsch+97] infrared-based system provides location information and messaging capability without the need of battery-powered tags and a network of sensors. It takes into account the privacy concerns associated with active badges by giving the user control of the location information and its release to the network. The Locust is dependent on a solar cell to provide its power and thus normally is placed in the grills beneath overhead fluorescent lights. Upon power-up the Locust begins broadcasting its location information. A user’s wearable computer can listen to this broadcast and decide whether or not to announce its location across the infrared link. The Locust Swarm’s advantage over the previously described infrared-based location systems is that it addresses privacy and decentralised location calculation. In contrast, it is much harder to

develop applications that require the location knowledge of other assets, such as computers or people.

2.1.1.5 Limitations of Infrared-based Location Technologies

The main limitation of the reviewed infrared-based location systems is the coarse location granularity they provide, *i.e.* the physical container within which they can be encountered. Whereas the Active Badge and Smart Badge systems require battery-powered, infrared emitting, tags to be attached to the objects that want to be located, the Locust Swarm system requires the mobile unit to have processing capability. As with any diffuse infrared system, these technologies have difficulty when deployed in locations with fluorescent lighting or direct sunlight because of the spurious infrared emissions these light sources generate.

2.1.2 Ultrasonic Technologies

2.1.2.1 Bat

The Bat [Ward98] ultrasonic indoor location system reports the position of objects (people and equipment) as the 3D coordinates in a frame of reference. Small battery-powered wireless units called *bats* are attached to equipment and carried by personnel. A bat consists of a radio transceiver, controlling logic containing a 48-bit globally unique identifier, and an ultrasonic transducer. Ultrasonic receiver units are networked at known points on the ceiling. A base station periodically transmits a radio message containing a single identifier, causing the corresponding bat to emit a short pulse of ultrasound. Simultaneously, the ultrasonic receivers in the room covered by the base station are reset via the wired network. Receivers monitor the incoming ultrasound and record its time of arrival. Using the speed of sound in air, the distances between bat and receivers are calculated. When distances from the bat to three or more non-collinear receivers are found, its position in 3D space is determined using a multi-lateration algorithm. The Bat system can also compute orientation information given predefined knowledge about the placement of two or more bats on the rigid form of an object. Coarse orientation from a single bat can also be obtained by analysing the location of the ceiling sensors receiving ultrasound pulses. Bats have also input and output facilities that take advantage of the bi-directional radio link.

The Bat system [Addlesee+01], shown in Figure 2.2, can locate bats to within 3 cm of their true position for 95% of measurements. Each base station can address three bats simultaneously, 50 times per second, given a maximum location update rate across each radio cell of 150 updates per second. The battery lifetime is 15 months.

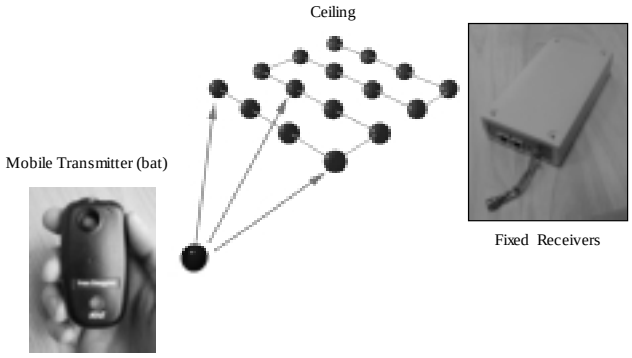


Figure 2.2. Bat Location System (courtesy of AT&T Labs Cambridge).

2.1.2.2 Cricket

The Cricket [Pryantha+00] location-support system also takes advantage of both ultrasonic and radio signals. Rather than the system tracking the user's location, each portable device must perform its own triangulation calculations. This system is conceptually similar to the Locust Swarm system. The mobile device listens to two signals (one RF and one ultrasound) that simultaneously originate from a base station and measures the distance to that base station based on the time interval between the arrivals of two signals. The resulting location accuracy is, in its current implementation, within 30 cm. Whilst the Bat system is a location-tracking system that stores location information for services and users in a centralised database, Cricket is simply a location-support system that provides the current location to the attached device or user. The advantages of Cricket include user privacy and decentralised scalability, while its disadvantages include a lack of centralised management or monitoring and the computational burden – and consequently power burden – that timing and processing both the ultrasonic pulses and RF data place on the mobile receivers. A similar location-support system providing even finer location granularity (10-25 cm) and lower cost ($150 of investment required per average room) is described in [Randell+01].

2.1.2.3 Limitations of Ultrasonic Location Technologies

The use of ultrasound time-of-flight as the method of measuring distance has the drawback of requiring a large fixed-transducer infrastructure throughout the ceiling and of being rather sensitive to the precise placement of the transducers. Thus difficulty of deployment, the use of custom-built hardware and cost (especially in the Bat case) are disadvantages of the ultrasonic solutions. These limitations prevent these systems from being adopted at homes, although they may be affordable and suitable for office spaces.

2.1.3 Radio-based Technologies

2.1.3.1 Passive RFID tags

A Radio Frequency Identification (RFID) tag is a digital processing unit, transmitter and induction power pickup loop. When exposed to an electromagnetic signal of the correct frequency the battery-less RFID tag charges and then transmits its identity number (usually on a different frequency) to an interrogator. Operating ranges are usually limited to within a few metres of the radio interrogator unit. The TI*RFID[4] (Texas Instruments Radio Frequency Identification Systems) is a good example of a battery-less RFID tag. TI*RFID tags are small (15mm long, 1mm wide) and low cost (< $2 each). Nevertheless, interrogators are expensive (around $500 each) for home or small office use. Moreover, TI*RFID focuses on object identification, only offering coarse location granularity, (bounded by the coverage area of a radio interrogator), and no object orientation details, due to radio's omni-directional propagation properties.

2.1.3.2 Active RFID tags

The 3D-iD [Werb+98] indoor radio-location system, based on proprietary L3RF (Low range, Long life, Low cost Radio Frequency) tags, can read signals at relatively long ranges (up to 50 metres) compared to conventional RFID systems while exhibiting a long battery life (6-12 months) at a reasonable cost ($10 each). These tags receive a low power 2.4 GHz spread spectrum radio signal from a cell controller or base station, and respond

[4] Formerly known as TIRIS tags (http://www.ti.com/tiris/docs/about/about.htm).

at defined intervals with 5.8 GHz signals that include identification data. A cell controller network continually tracks 3D-iD signals. Each cell controller can consist of up to 16 antennae that receive signals back from the 3D-iD tags. By calculating the round trip times of signals detected by its antennae, a 3D-iD cell controller can identify the location of the tag to an accuracy of around 1-3 metres. The 3D-iD system has the disadvantage that each radio directional antenna has a narrow cone of influence, which can make ubiquitous deployment prohibitively expensive. Thus, 3D-iD best suits large indoor space environments such as hospitals or warehouses. The coverage range of 3D-iD antennae is much bigger than with TI*RFID, but in this case the tags must be powered, and the system cost is also much higher ($10,000 for an eight-antenna kit).

2.1.3.3 WaveLAN-based Location Systems

The RADAR [Bahl00] system is a building-wide tracking system based on the IEEE 802.11 WaveLAN wireless networking technology. RADAR measures at the base station the signal strength and signal-to-noise ratio of signals sent by wireless devices. It then uses this data to compute these devices' two-dimensional (2D) position within a building. The location is determined by querying a central database of RF signal strength at a set of fixed receivers for known transmitter positions. The database with RF signal strength measurements is created during a system calibration stage. Similarly, the Nibble system [Castro+01] measures the signal to noise ratio (SNR) between the transmitter and different base stations, while a pre-encoded Bayesian network model describes the joint probability distribution for location, given base-station SNRs. Both Nibble and RADAR present two advantages: (1) they require relatively few base stations and (2) they share the same infrastructure that provides the building's general-purpose wireless networking. Likewise, they share three disadvantages: (1) only objects supporting a wireless LAN are trackable, (2) generalising those systems to the 3D dimensions or buildings with multiple floors is a nontrivial problem, due to the omni-directional nature of radio propagation and (3) the wireless LAN cards currently require high battery power.

2.1.3.4 Limitations of Radio Indoor Location Technologies

The main drawback of the radio-based indoor location technologies described is that the topological divisions of buildings, *i.e.* rooms or floors, are not easily detected due to the propagation characteristics of radio. For instance, if a user is looking for the "closest

printer", he really means the one in the same room and not the one on the other side of a wall, despite the fact that the latter one may be closer in absolute distance.

2.1.3.5 GPS

The Global Positioning System (GPS) [Trimble01], without a doubt the most widely available location-sensing system, is an outdoor positioning system created by the US Department of Defence. Aircraft, hikers, search-and-rescue teams and rental cars use this positioning system. GPS is based around 24 satellites that circle the Earth twice a day in very precise orbits and transmit spread-spectrum radio signals. GPS receivers take this information and use triangulation to calculate the user's exact location. Essentially, every GPS receiver, see Figure 2.3, compares the time a signal was transmitted by a satellite with the time it was received. The time difference tells the GPS receiver how far away the satellite is, using the speed of light through the atmosphere. With four or more satellites in view, the receiver can determine the user's 3D position (latitude, longitude and altitude).

Figure 2.3. GPS handheld receiver[5].

Since the US Government's removal of the Selective Availability (SA) degradation [Dana00], standard GPS receivers are accurate to within 15 metres on average. Newer GPS receivers with WAAS (Wide Area Augmentation System) capability can improve accuracy to less than three metres on average. No additional equipment or fees are required to take advantage of WAAS. Unfortunately, WAAS is only available in USA.

[5] Image copyright GARMIN.

The European Space Agency plans to make the European Geo-stationary Navigation Service (EGNOS) available in 2004. This will provide similar levels of location accuracy to WAAS for the European continent. Users can also get better accuracy with Differential GPS (DGPS), which corrects GPS signals to within an average of 3-5 meters. This system consists of a network of towers at known locations that receive GPS signals and transmit a corrected signal by beacon transmitters. In order to get the corrected signal, users must have a differential beacon receiver in addition to their GPS.

The main limitation of GPS is that it cannot be used indoors, because the frequencies at which the satellites transmit signals do not penetrate buildings. Interestingly, some companies [Parthus01] are currently making progress towards enabling GPS to be used indoors. However, the maximum three metre accuracy provided by GPS will not be sufficient to develop useful Sentient Computing applications in an indoor environment.

2.1.3.6 Mobile Phone-based Location Technologies

Since the U.S. Federal Communications Commission mandated that all calls from mobile phones to emergency services should include information about the caller's location (within 150 metres), several location systems have been proposed for wide-area cellular systems [Tekinay98]. The technological alternatives for locating cellular telephones [Andersson+01] involve measuring the signal attenuation, the angle of arrival (AOA), and/or the time difference of arrival (TDOA).

2.1.3.7 Limitations of Outdoor Radio Location Technologies

While these systems have been found to be promising in outdoor environments, their effectiveness in indoor environments is limited by the multiple reflections suffered by the RF signal, and the inability of off-the-shelf and inexpensive hardware to provide fine-grained time synchronisation.

2.1.4 Technologies based on Ground Force

The Active Floor [Addlesee+97], and its later clone the Smart Floor [Orr+00], try to identify people using the vertical component of the Ground Reaction Force (GRF). This unobtrusive direct physical contact system does not require people to carry a device or

wear a tag. Nevertheless, it can only track the location of objects on top of it. Headon [Headon+01] is extending the Active Floor system to permit not only the identification and tracking of people but also the recognition and characterisation of some of their functional movements, such as when the user is taking a step, jumping, bending his back or sitting down. This work is of special appeal for the prevention of injuries on people undertaking tasks that require physical effort, *e.g.* weight lifting. The main drawback of GRF-based technology for location tracking is its poor scalability and high incremental cost, since the floor of each building in which the system is deployed must be physically altered to install pressure sensor grids. This cost is not affordable in home environments but it may be possible in commercial office environments.

2.1.5 Magnetic Field-based Technologies

Magnetic trackers use the reduction in the strength of a magnetic field to measure accurately the six-degrees of freedom (*i.e.* the translation vector and three orientation angles defining a 3D position) of a moving object. These tracking systems generate axial DC magnetic-field pulses from a transmitting antenna in a fixed location. The system computes the position and orientation of the receiving antennae by measuring the response in three orthogonal axes to the transmitted field pulse. A good example of these systems is the Flock of Birds [Ascension01] system. These systems offer the advantage of very high precision and accuracy, on the order of less than 1mm spatial resolution, 1ms time resolution, and 0.1° orientation capability. Disadvantages include their high cost (from $1,500 to $90,000) and the need to tether the tracked object to a control unit. Further, the sensor must remain within 1-3 metres of the transmitter, and accuracy degrades with the presence of metallic objects in the environment. These technologies are mainly used for virtual reality environments or to support computer animation.

2.1.6 Vision-based Technologies

2.1.6.1 Pfinder

The Pfinder (*person finder*) [Wren+97] tracker applies sophisticated computer vision techniques to recognise the presence of a user in the environment without them needing to wear any special marker. Motion in video images is used to identify the presence of a person and subsequently the system uses a multi-class statistical model of colour and

shape to obtain a 2D representation of head and hands in a wide range of viewing conditions. Unfortunately, many objects of interest (*e.g.* a laptop computer) are less mobile and distinctive than people. It is also difficult to determine the identity of the person tracked by the system. To make the vision task tractable, Pfinder expects the scene to be significantly less dynamic than the user and that only one user is in the recognition space.

2.1.6.2 Tracking Multiple People with Multiple Cameras

[Stillman+99] describes a real-time method for tracking and recognising multiple people with multiple cameras. They use both static and pan/tilt/zoom (PTZ) cameras to provide visual attention. The PTZ camera system uses face recognition to register people in the scene and "locks on" to those individuals. The static camera system provides a global view of the environment and is used to re-adjust the tracking of the system when the PTZ cameras lose their targets. The underlying visual processes rely on colour segmentation, movement tracking and shape information to locate target candidates. Colour indexing and face recognition modules help register these candidates with the system. The system can reliably track and locate up to two people in the viewing area. Users are required to train the system to recognise their faces. Unfortunately, the processing requirements of the system, two SGI Indy R5000 workstations and two Pentium II PCs, and expensive cameras (SONY EVI-D30 PTZ at around $1,200 each) make this system unsuitable for wide deployment.

2.1.6.3 EasyLiving Vision Module

The EasyLiving project's vision module [Krumm+00] uses two Digiclops cameras (see Figure 2.4), producing real-time range images using stereo computer vision technology, to track personnel location and objects within a living room environment. Using the registered depth and colour images provided by the cameras, background subtraction is performed to locate 3D blobs in each camera's field of view. These blobs, normally broken up over the regions of peoples' bodies, are merged into person shapes by examining the space of blob clustering. The Stereo Module uses stereo cameras to locate people-shaped blobs. Each Stereo Module reports the 2D ground plane location of its person-shaped blobs to a tracking program, called the Person Tracker. The Person Tracker uses knowledge of the two cameras' relative locations, fields of view, and

heuristics on the movement of people to produce a final report on the locations and identities of people in a room. Colour histograms of the person-shaped blobs are used to disambiguate when people are close together.

Figure 2.4. Digiclops stereo video camera[6].

The EasyLiving person tracking system requires three state-of-the-art PCs to accurately locate (within 5 cm) a maximum of three people in the field of view of the cameras, at 7Hz. The system cannot infer the identity of individuals; it can only track a colour histogram, to which a virtual identity is associated, throughout the room. The Digiclops high-performance cameras perform part of the stereo computation in hardware, significantly reducing the amounts of processing power required to analyse frames on the PCs. Nevertheless, these cameras are very expensive ($6,000 each), and although the system is completely unobtrusive (the user does not have to wear a tag), its reliability, due to occlusions, is much worse than other tagged indoor location systems such as the Bat. In addition, users may feel uneasy about the possibility of their whereabouts being recorded by the deployed cameras.

2.1.6.4 Limitations of Untagged Vision-based Location Systems

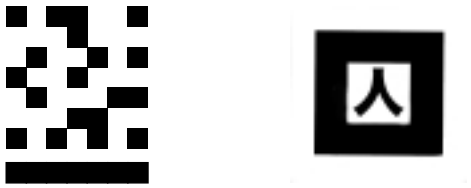
Vision-based systems have difficulties in maintaining analysis accuracy as scene complexity increases and more occlusive motion occurs. Moreover, untagged solutions rely on sophisticated image processing and frame correlation techniques, requiring a lot of CPU processing power. Furthermore, objects distinguished from the scene background (using colour histograms, for example) can be accurately located using geometric algebra

[6] Image copyright Point Grey Research.

if more than one camera is available, but identifying the object, like face recognition [Kruizinga01], remains a hard problem.

2.1.6.5 CyberCode

The CyberCode [Rekimoto98][Rekimoto+00] system proposes a method of identifying real-world objects and estimating their positions and orientations using a combination of visual markers and a video camera. A CyberCode tag (see Figure 2.5a) is a square 2D matrix marker, where information is encoded as black and white square cells within the matrix. The black guide bar of the CyberCode facilitates the identification of the four corners of the ID encoding matrix within an image. A tag represents a 24 bit binary number, thus allowing a large number of objects to be tagged. By analysing the distortion of the square shape of the matrix (of known size), the system estimates the position and orientation of the video camera and determines the transformation matrix between the real-world points (expressed in a marker-centred coordinate system), and the image points.



a) CyberCode Tag b) ARToolkit Marker

Figure 2.5. 2D square markers[7].

2.1.6.6 ARToolKit

The ARToolKit [Kato+99] system also uses a square marker (see Figure 2.5b) to extract the location and orientation of tagged objects, by applying similar computer vision techniques as CyberCode. However, its identifier encoding capabilities are more limited; the sub-image within the black square frame marker used to encode an identity has to be compared by template matching with patterns that are previously specified to the system. The CyberCode and ARToolKit systems have mainly been applied to produce AR

[7] Image copyright for ARToolKit tag owned by HIT Lab, University of Washington.

(Augmented Reality) [Azuma97] applications, *i.e.* to register computer-synthesised information on views of the real world.

2.1.6.7 Limitations of Tagged Vision-Based Location Systems

The markers used for CyberCode and ARToolKit are designed to extract the 3D location and orientation of objects relative to the cameras with high accuracy, within 1 cm and 1 degree on average, and to simplify the computational complexity of the computer vision task. Thus, both technologies can operate in real-time. Nevertheless, their markers' geometric features require high image resolution for accurate recognition, *i.e.* the projections of markers have usually to occupy areas of more than 100x100 pixels in the image. This factor prevents these technologies from being used as generic object tracking mechanisms. Further, the square-based markers are difficult to identify in cluttered environments, which contain many square shaped objects (*e.g.* monitors, tables, etc.). Finally, these vision-based location technologies are subject to the *line-of-sight* constraint and the relatively high computational demands typical of visual systems.

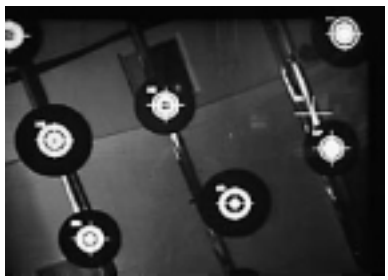


Figure 2.6. 2D circular markers[8].

2.1.6.8 Tagged Hardware-Implemented Vision Location Systems

The BBC's Free-D [Thomas+97] camera location system measures the precise position and orientation of studio cameras by using an auxiliary camera mounted on the back of a conventional moving camera pointing to 2D circular markers fixed on the ceiling of a TV recording studio (see Figure 2.6). Real-time video processing is achieved through a hardware implementation of the computer vision algorithms. The system is used for virtual reality TV production, producing a very high location and orientation accuracy

[8] Image copyright BBC Research & Development.

(within 1 mm and 1 degree error). Free-D is much more expensive and cumbersome to deploy than CyberCode and ARToolKit systems, although it identifies targets located at a further distance, *i.e.* it requires smaller pixel areas for accurate target recognition.

2.1.7 Sensor Fusion for Location Sensing

As shown by the previous sections, there is no uniform way to track location with fine granularity that works well both indoors and outdoors. In practice, a system may have to consult different location systems to locate different objects. Each kind of sensed data has an associated uncertainty due to environmental noise or sensor errors. Even worse, context sensed from different sensors may conflict.

Sensor Fusion [Brooks+98] is a technique that can be employed to solve such conflicts and improve context accuracy and completeness. Defined as the simultaneous use of multiple sensing technologies to form hierarchical and overlapping levels of sensing, sensor fusion can provide additional properties unavailable when using the sensing devices separately. Rizzo *et al.* [Rizzo+94] propose a Master Location System as a framework to allow multiple location mechanisms to co-exist and co-operate. Capability-based access control and several different location source corroboration and conflict resolution functions are discussed. A homogenous form of representing location, useful when fusing location data, which considers the uncertainty inherent in each sensing technology, is given in [Leonhardt98].

2.1.8 Review of Location Technologies

Table 2.1 summarises the features for the location technologies described in this chapter. From this study, it can be concluded that location technologies for indoor environments have failed to reach the mass market principally because of their bespoke hardware requirements, complex installation and high cost. Likewise, these factors have also prevented a wider application of the Sentient Computing paradigm in our living and working environments. Therefore, there is a need for the development of a new location technology accessible to everyone and that can be installed anywhere, *e.g.* at home or in the workplace. The requirements such a ‘location sensor for the masses’ presents are:

- Use commonly available, off-the-shelf, technology.
- Require low maintenance costs and non-technical expertise.
- Provide fine-grained location information at a high update rate.
- Be minimally obtrusive.

From the plethora of technologies reviewed none completely fulfils these requirements. The CyberCode [Rekimoto+00] and ARToolKit [Kato+99] tagged software-based vision location technologies most closely approach these requirements. Nevertheless, the main problem with these systems is the necessity for the objects to be located to be within a close distance (one metre or less) of the viewing camera. Chapters 3 and 4 of this dissertation describe a tagged, vision-based, low-cost and easily deployable, software-based 3D location technology that fulfils the above requirements, and demands a lower camera resolution than the Cybercode or ARToolKit systems.

Technology	Physical Medium	Tagged (Active/Passive)/ Untagged	Ident./ Orien.	Physical/ Symbolic	Centralised / Local Location Computation	Location Granularity / Frequency	Off-the-shelf/ Bespoke Hardware Requirements	Cost	Limitations
Active Badge	Infrared	Tagged (Active)	I	S: room ID	Centralised	Room size, once every 10 secs	B: infrared sensors + badges	Maintenance costs	Sunlight and fluorescent light interference, thick tags (1 cm)
Bat	Ultrasonic	Tagged (Active)	I/O?	P: 3D coordinates	Centralised	3 cm (95% cases), 150 Hz per room	B: ultrasound sensor per ceiling tile + badges	Maintenance costs, expensive ($1,500 per room)	Very cumbersome to deploy. Thick (1.5 cm) tags
Cricket	Ultrasonic	Tagged (Active)	I	PS: knows distance to given base station	Local	30 cm (98% cases)	B: ultrasound beacons + receivers	$10 each beacon and receiver.	No central management, receiver computation
PinPoint's 3D-iD	RF	Tagged (Active)	I	P: distance to base station	Centralised	1-3 metres	B: base stations + tags	Expensive ($10,000 for 8-antenna kit)	Complex infrastructure
RADAR/ Nibble	802.11 RF	Tagged (waveLan card)	I	P: distance to base station	Centralised	3-4.3 metres (50% cases)	O: Wavelan Base Stations + PC cards	802.11 LAN installation + $100 per wireless NIC	Only attachable to devices with wireless NIC
GPS	Radio	Tagged (Active)	None	P: latitude, altitude, longitude	Local	1-5 metres (95-99% cases)	O: only requires purchase of GPS unit.	$100 per GPS receiver	Does not work indoors
Active Floor	Weight cells	Untagged	I?/O	P: exact position within floor	Centralised	Locates centre of mass (1cm), very fast (1KHz).	B: specially designed floor sensor grid.	Data acquisition card is expensive ($1,000)	Expensive, difficult to infer identity
Flock of Birds	Magnetic	Tagged (Active)	I/O	P: 6 degrees of freedom	Centralised	1 mm, 1ms, 0.1° (nearly 100 %), 100 Hz	B: specially build controller and sensors	Expensive hardware (from $1,500 to $90,000)	Control unit tethered, precise installation, high cost
EasyLiving	Vision	Untagged	I?/O	P: distance and orientation with respect to camera	Centralised	Within 5 cm, 7Hz	O: PCs+cameras	Uses very expensive cameras ($6,000)	Line of sight, high processing power, no identification, tolerates few people in room
CyberCode / ARToolKit	Vision	Tagged (Passive)	I/O	P: distance and orientation with respect to camera	Centralised	Within 1 cm, 1 degrees of error, 20 Hz	O: PCs+cameras	Very low, printed tags + cheap CCD cameras (around $50 each)	Line of sight, relatively high processing power, requires objects to be very close to camera
TRIP (see chapters 3 and 4)	Vision	Tagged (Passive)	I/O	P: distance and orientation with respect to camera	Centralised	3% average error on location and 2% on orientation, 16 Hz	O: PCs+cameras	Very low, printed tags + cheap CCD cameras (around $50 each)	Line of sight, relatively high processing power

Table 2.1. Comparison of location technologies.

2.2 Management of Context Information

The sheer diversity of exploitable contexts and the plethora of sensing technologies are actually working against the deployment of context-aware systems, as suggested by [Pascoe98]. Context is difficult to use by applications [Dey+00] for the following reasons: (1) context is often acquired from non-traditional devices, (2) context must be abstracted to make sense to applications, (3) context must be acquired from multiple distributed and heterogeneous sources and (4) context is dynamic, *i.e.* changes in the environment must be detected in real-time and applications must adapt to those changes. In essence, the large development effort required for Sentient Computing has prevented a more widespread adoption and experimentation of this computing paradigm.

Therefore, it seems sensible to decouple the application and the actual sensing of context, because of the large development overhead of interacting with a variety of sensors to capture the context, converting it to the desired format, and presenting it in a meaningful way. Software that supports the collection of raw sensor information, its transformation into an application-understandable format, and its dissemination to interested applications is necessary. In what follows, a set of projects providing software support for Sentient Computing are considered. The software support these systems provide is categorised as being in the form of libraries, frameworks, toolkits or services. In order to understand the implications of such different kinds of support, a definition of each term is given:

- A *library* is a generalised set of related algorithms, *e.g.* string handling, focused exclusively on code reuse.
- *Framework*s concentrate more on design reuse by providing a basic structure for a certain class of applications.
- *Toolkit*s build on frameworks by also offering a large number of reusable components for common functionality.
- An infrastructure is a well-established, pervasive, reliable, and publicly available set of technologies that act as a foundation for other systems. *Service infrastructures* are middleware technologies that can be accessed through the network.

2.2.1 The Active Badge Distributed Location Service

A scalable indoor Location Service to manage the location data provided by Active Badges is proposed in [Harter+94]. A centralised Location Server maintains a cache of the most recent piece of location information for every badge detected. The location unit kept for each badge consists of a badge address, a location and a timestamp. Interfaces are provided for clients to invoke person- or location-centric queries. These interfaces are suitable for clients that start, interact with the system and terminate. For long-living clients a publish-subscribe interface to enable the specification of a filter and a call-back is provided. In this way the Location Service delivers only changes in location that are of interest for the registered client. A Directory Service that provides applications with lookups by name or by address for badges, equipment, locations and domains is also provided.

A system analogous to the Active Badge Location Service, named Active Map Server (AMS), is presented in [Schilit+94b]. This system differs in the way location-based messages are disseminated to interested clients. The AMS recognises when multiple clients are specifying the same subscription query, by forcing clients to use given query templates, and employs a *multicast* channel to service the update traffic for that query. In essence, the AMS approach is to use large numbers of multicast groups in order to keep client filtering overhead and slow communication link loads to a minimum. Both the Active Badge Location Service and AMS are accessible to clients through libraries.

2.2.2 The Situated Computing Service

The above-mentioned Location Services focus on location data collection and distribution, leaving the sensor data interpretation task to applications. Moreover, these services are intrinsically tied to their underlying tracking technologies, namely the Active Badge and the PARCTab, respectively. The Situated Computing Service (*SitComp*) [Hull+97] is an attempt to solve the limitations of these services. SitComp interprets sensor data coming from different context-sensing technologies (not just location technologies) and provides context-aware information at an appropriate level of abstraction for applications. Query and notification interfaces are provided to communicate the current situation to applications.

The SitComp server is composed of a dynamic network of connections between sensors, interpreters, and applications waiting for a situation matching. As sensor data flows up through this network, this information is combined and abstracted by interpretation layers until the dataflow reaches a level of abstraction appropriate to interested applications and an event is posted. All entities in the network join by registering as producers or consumers of situational information. Data fusion and abstraction is achieved by enforcing standard formats for situational dimensions, and ensuring that the output of all entities sourcing a dimension are routed to an appropriate interpreter. The main limitation of SitComp is that clients can only register to receive notifications generated by available interpreters. If there is no interpreter providing the contextual situation demanded, a new interpreter has to be provided by the application developer and the SitComp server must be modified.

2.2.3 SPIRIT

The SPIRIT[9] [Adly+97] project aims to support mobile users who move around in an office environment without undue degradation in the computing and communication resources available to them. To achieve this purpose, information about the environment is gathered from sensor sources, including the Active Badge, the Bat and telemetry software monitors for keyboard, CPU, disk and network activity. The resulting data is combined with static data about the building, people and equipment, to create a detailed model of the environment. This model sets up the types, names, capabilities and properties of all entities (people, computers, telephones, etc.). The software counterparts of real-world entities are implemented as persistent distributed objects using CORBA and an Oracle 7 database. These persistent objects provide information to mobile applications via query and call-back interfaces and are accessed through a proxy server.

The SPIRIT project observes the interest of location-aware applications in relative spatial facts rather than absolute ones, *e.g.* the Bat system would provide the fact ‘the person is at x, y, z, facing in direction θ’ whereas applications are interested in ‘the person is

[9] The name of this project has been changed to “Sentient Computing”. However, in order to distinguish this project from the research topic itself, the old name is used.

standing in front of the workstation'. To address this issue, SPIRIT defines a real-time *Spatial Monitoring Service* [Harter+99] that expresses relative spatial facts about objects in terms of geometric containment and overlapping relationships between spaces associated with those objects. Location events generated by moving objects are used as input for an indexing system, which calculates all containment and overlapping relationships and broadcasts them to registered applications using a performance-tuned event service. Spatial Monitoring [Addlesee+01] replaces human computer interaction (through mouse or keyboard) by allowing users to interact directly with space.

The main criticism of this work is the almost exclusive focus on the location attribute of context (apart from the monitoring of computing resources) for the development of sentient systems. In addition, the implementation of the Spatial Monitoring Service can only handle location information fed from the sophisticated fine-grained Bat location system. Other researchers have addressed this limitation by developing a similar Spatial Monitoring Service capable of combining information from diverse location technologies and with different confidence levels [Naguib+01][Leonhardt98].

2.2.4 The Stick-e Note Architecture

The development of context-aware applications is complex, requiring the skills of highly capable systems programmers. This factor has prevented a wider transition of context-aware applications from the research laboratory to the marketplace. Based on this observation, [Brown+97] propose a software infrastructure, the *stick-e note*, that makes the creation of *discrete* context-aware applications as easy as producing a Web document. In discrete context-aware applications, separate pieces of information are attached to individual contexts (rooms, time ranges, being with certain people) that are triggered when the user enters those contexts. This infrastructure is targeted to a scenario in which the mobile user carries a Personal Digital Assistant (PDA) with environmental sensors attached which sense aspects like location, orientation and temperature.

A stick-e note [Brown96] is an electronic equivalent of a Post-it note associated with a particular context. SGML (Standard Generalised Mark-up Language) is chosen to make the exchange and publish process easy. A DTD (Document Type Definition) specification defines all the mark-up tags that can be used for writing rules. These rules specify what

actions to take when a particular combination of contexts (*i.e.* a situation) occurs. A repository of stick-e notes resides either on the user's PDA or on a backbone network server from which contexts and notes can be transferred using a wireless link. Authors create notes associated with a given context and a general-purpose triggering engine activates these when they match the user's present context.

The mandatory use of notes by clients of context information makes it difficult to retrofit an existing application with context awareness or even build an application that modifies its behaviour in response to a changing environment. The approach of the stick-e note framework is very simple but quite limited as a result of their efforts to support non-programmers.

2.2.5 EasyLiving

EasyLiving [Brumitt+00] is a research project investigating appropriate architectures and technologies for intelligent living environments. The goal of this project is to develop an architecture that will support a coherent user experience as users interact with a variety of devices in the intelligent environment. Software components are used to represent and encapsulate input and output devices. They register with a lookup service to allow other components and applications to locate them when necessary. A middleware framework called InConcert is used to support communication between distributed components, using asynchronous messages constructed in XML. A major focus of the EasyLiving project is to accurately model the geometry of the intelligent environment to better inform applications of physical relationships between people and devices or locations. This approach is very similar to that introduced by the SPIRIT project. The architecture developed for EasyLiving uses InConcert and individual software components to provide support for separation of concerns, transparent communications and constant availability of components to multiple applications.

2.2.6 CoolTown

CoolTown is an infrastructure that supports context-aware applications by representing each real-world object, including people, places and devices, with a web page [Caswell+00][Kindberg+00]. Each web page dynamically updates itself as it gathers new

information about the entity it represents. The main motivation of this project is to give users access to devices that augment the user's reality with web services related to the physical objects they see. For example, as users move through an environment, they will see a list of available services for this environment. They can request additional information or can execute one of the services. The CoolTown infrastructure provides abstraction components (URLs for sensed information and web pages for entities) and a discovery mechanism to make the building of these types of applications easier. Since it is focused on displaying context, it is not intended to support other context-aware features, such as automatically executing a service based on context or tagging captured information with context for later retrieval.

2.2.7 The Context Toolkit

The goal of the Context Toolkit [Salber+99][Dey+00] project is to construct a framework that makes it easier to design, build and evolve context-aware applications. This framework aims to have application designers expend less effort on the details that are common to all context-aware applications and focus on the main goal of these applications: specifying the context their applications need and the context-aware behaviours to be implemented. The Context Toolkit architecture consists of three different types of abstractions named *widgets, aggregators* and *interpreters*, upon which applications can be modelled. *Context widgets* are responsible for acquiring a certain type of context information and making it available to applications in a generic manner, regardless of how it is actually sensed. Applications can either query the state of a widget or register specifying the context notifications in which they are interested. *Context aggregators* gather the context about an entity from the available context widgets and aggregate it, acting as a context-widget proxy for the final context-aware applications. *Context interpreters* are used to abstract or interpret context. For example, a context widget representing a GPS receiver may provide location context in the form of latitude and longitude, but an application may require the location in the form of a street name. It is the job of the interpreter to translate between different ways of expressing context.

Context components are instantiated and executed independently of each other in separate processes and on separate computing devices. The communication between applications and components is supported via XML messages using HTTP. Applications can search

for the properties and network location of context widgets through a centralised Context Discoverer component. It is argued that this architecture makes the development of context-aware applications as simple as developing GUI applications. In order to foster the building of applications that use a variety of sensors and contexts and that are easy to change, it addresses the following requirements: (1) separation of concerns between context acquisition and context use, (2) notification about relevant changes to context values, (3) interpretation and storage of context. Nevertheless, this architecture does not address efficient context dissemination, since HTTP is not a good protocol for transmitting events. Moreover, it does not provide support for generic context aggregation, *i.e.* programmers must develop a different context aggregator for each situation that has to be monitored, *e.g.* "people in a meeting" context aggregator.

2.2.8 Context Fabric

The Context Fabric project [Hong+01] focuses on providing service infrastructure support for context-awareness in order to simplify the task of creating and maintaining context-aware applications. Context Fabric shifts as much of the weight as possible of context-aware computing onto network accessible middleware infrastructure. Application development is assisted by a set of uniform abstractions and reliable services for common operations. The burden of acquisition, processing, and interoperability is placed on network accessible infrastructure instead of on individual devices and applications. Two basic infrastructure services proposed by the Context Fabric are: (1) Automatic Path Creation service, simplifying the task of refining and transforming raw sensor data into higher-level context data, and (2) Proximity-Based Discovery service which enables a device to request that the infrastructure locates the low-level sensors at a given location. It is argued that a system infrastructure approach offers the following benefits in comparison with libraries, frameworks and toolkits: (1) independence from hardware, operating system and programming language, (2) improved capabilities for maintenance and evolution and (3) sharing of sensors, processing power, data and services, making applications easier to develop and deploy. On the other hand, a system infrastructure approach may incur in higher network consumption and the common single point of failure/bottleneck problem associated with centralised systems.

2.2.9 Context Management Review

In this section a set of solutions addressing the difficult task of handling context information have been described. Starting from the most basic architectures providing support for Sentient Computing development, this section has progressed into more involved framework-, toolkit- and network service- based platforms. This dissertation shares the view with the Context Fabric project in that there are several benefits to having a service infrastructure for context awareness over having just a library, framework, or toolkit support. The main advantage is that the application developer can simply delegate the context capture and interpretation duties to network-based services, concentrating solely on the handling of notified contextual situations. This way, sentient application development is reduced to the specification of the context needs and the implementation of the reactive behaviour desired. Chapters 5, 6 and 7 cover the work of this dissertation in this area. The solutions presented in those chapters are influenced by concepts put forward by the Situated Computing, Context Toolkit and Context Fabric systems.

2.3 Applications

In this section, an overview of the most relevant context-aware applications is given. The first catalogued context-aware applications were produced using the Active Badge location data. In the first paper [Want+92] describing this technology, a telephone call routing application is presented that facilitates redirection of telephone calls to the phone closest to the target individual. Later, Active Badge data [Harter+94] was used to provide hands free access control to workstations and doors and to define a ‘nearest-printer’ service, offered to users of portable computers, which automatically directs the print command output to the closest printer.

The Active Badge System has also been used for mobilising user applications. The Teleporting System [Richardson94] applies the location information provided by personnel and equipment Active Badges and the control inputs resulting from pressing the Active Badge’s buttons to reallocate a user’s desktop to her new location. In [Bates+96], a framework for building location-oriented multimedia applications is described which enables multimedia objects to follow the user.

The PARTab device [Schilit+94a] has been utilised in the implementation of a variety of applications involving automatic contextual reconfiguration and context-triggered actions. A multi-user drawing program built to provide a virtual whiteboard for a room causes an automatic binding between a mobile host entering a room and the virtual whiteboard. A contextual reminder application is produced that enables the specification of when a reminder should occur. When the context specified is matched, a reminder note is triggered. The PARCTab technology has also been used in the implementation of the Forget-me-not system [Lamming+94], a context-aware based retrieval application whose function is to act as an *aide-mémoire*. This system aims to have a large number of sensors around the workplace in order to capture as much as possible about the user's working life: what rooms they were in, who they were with, what communications they sent and received, and so on. This information is then available whenever the user wants to recall past situations. This project demonstrates that a user's context can provide a valuable key for automatic indexing of information. Context-aware retrieval [Brown+01] appears to be a current popular research topic in the area of context-aware computing.

The Smart Badge [Beadle+98] technology has been used to prototype a Smart Hospital application that aims to improve information flow in a hospital. A central computer stores hospital records and provides authentication and access control services. Patients are equipped with Smart Badges that report their current location and monitor temperature, respiration and heart rate. The doctor carries a Smart Badge attached to a wireless communication-enabled PDA where patient records for nearby patients are automatically displayed. Results of each examination performed on the patient are entered by the doctor into her PDA and made available through a wireless link to the central Smart Hospital server. The input of badges worn by doctors is also used to provide access to restricted areas. A very similar application is also prototyped by the QoSDream project [Mitchell+00]. This project addresses the requirements for immediate high-quality multimedia communications in environments where users work practices exhibit a large degree of physical mobility, *e.g.* a hospital. By integrating a multimedia framework with an event-based notification system, a platform is provided that can offer seamless, context-sensitive communications, which can adapt to users location and even follow them around.

The CyberCode and ARToolKit barcode-based location technologies are mainly used to produce applications in the AR domain. For example, CyberCode [Rekimoto+95] has been used to augment the view of a museum visitor through a see-through device with personalised computer synthesised information according to the user's age, knowledge level and preferred language. [Rekimoto+00] offers a good overview of other applications produced with CyberCode, e.g. the launching of an application by showing a CyberCode to a camera. The ARToolKit [Billinghurst+99] has been used to create an AR conferencing system which overlays virtual images on the real-world view of a user wearing a see-through head mounted display. Remote collaborators are represented as live video images, which are attached to tangible objects that can be freely positioned about a user in space. It is argued that the use of AR overcomes some of the limitations associated with traditional video conferencing and allows the user to conference from anywhere in their physical environment.

The Bat system design motivation was to enable new kinds of location-aware applications only possible with higher location resolution. A limitation of the Teleporting System [Richardson94] is that when a user clicks an Active Badge button, their desktop is teleported to one of the several screens available in the room, but not necessarily to the closest one. [Harter+99] explains how the SPIRIT system combines precise Bat location and coarse orientation with resource monitoring information to allow the redirection of a user's desktop to the closest non-utilised display. Likewise, in [Ward98] a walk-through videophone application is described that enables a nomadic user to carry on a videoconference; the system monitors the user location and orientation accurately in relation to the available video and sound input and output resources.

The stick-e notes infrastructure has been used to produce PDA-based context-aware applications. [Brown+97] describes an application that uses stick-e notes to cover paging requests. Somebody looking for a book she cannot find can create a stick-e note expressing her wish to obtain it. Whenever somebody's PDA can sense the presence of such book, a paging message is triggered indicating that somebody else is looking for it. In [Pascoe98], an application is described applying the stick-e notes technology in ecological fieldwork. An ecologist was provided with a PDA with an attached DGPS unit and a stick-e note infrastructure based application to assist her in the data collection task to investigate the feeding behaviour of giraffes. The context associated with each note

was automatically captured by the system (time and location) letting the user concentrate on her observational tasks. In addition, buttons of her PDA were customised to serve as input for her observations, without the user needing eye contact with the device.

The Context Toolkit architecture is used in [Dey+99] to create a Conference Assistant. This application tries to help conference attendees to: (1) decide which activities to attend, (2) provide awareness of the activities of colleagues, (3) assist users in taking notes on presentations and (4) aid in the retrieval of conference information after the conference concludes. It uses a variety of contextual data: attendee's location, current time and schedule of presentations. Furthermore, it combines in one application most of the features provided by other context-aware applications: (1) it presents information and services to the user (conference timetable and which colleagues are attending which conference); (2) it automatically executes services (it automatically updates the current slide in the user PDA); and (3) it tags current context to information for later retrieval (notes made by the user are augmented with contextual information). Conference attendees execute the application in PDAs with 3D-iD RF-tags to obtain location provided by the conference organisation. Context-based retrieval is also possible from the attendee's home location to revise the material presented in the conference.

The CyberGuide [Long+96] and GUIDE [Davies+01] systems are examples of context-sensitive tour guides. The goal of these systems is to provide tailor-made tours for visitors by adapting their behaviour to changes in a user's location and context. These systems generate a different tour depending on a visitor's interests, location, available time, mobility constraints, or local weather conditions. CyberGuide and GUIDE provide visitors to the Georgia Tech campus and the city of Lancaster, respectively, with context-sensitive information presented via an electronic handheld device. CyberGuide uses an infrared-based location system indoors and GPS outdoors. GUIDE uses location identifier beacons broadcast by WaveLAN base stations.

Probably, city or campus guides like the ones mentioned or car navigation systems are the type of context-aware applications experiencing a bigger commercial success. Marketing surveys on the upcoming 3G wireless communication systems [Andersson+01] show that there is one feature of the mobile Internet that users are keen to start using: location-based

services. Thus, personalisation of services to mobile phone users location (provided by UMTS) and context may lead to the commercial success of Sentient Computing.

2.4 Acceptability of Sentient Computing

An important reason that some may use to argue against the promotion of the Sentient Computing paradigm is the set of social issues it raises. In particular, security and privacy of the context information associated with a user are paramount. Most people do not like the idea of being precisely located at any time, by anyone, especially when the location data is logged. Therefore, it is important to consider privacy issues in context-aware computing. Furthermore, it is prudent, to stress the potential intrusiveness associated with Sentient Computing activities. This dissertation does not cover such aspects. However, other authors have provided solutions for these problems, *e.g.* [Spreitzer+93], [Jackson96], [McCandless97] and [Rizzo+94]. This work attempts to prove that usually the benefits from adopting this computing paradigm outweigh its intrinsic drawbacks in the areas of security, privacy and intrusiveness. Mobile telephones and credit cards also present some of these problems but are still widely embraced since they provide tangible benefits to the user. The same should apply to the use of Sentient Computing.

2.5 Conclusion

This chapter has reviewed the state-of-the-art in Sentient Computing research. First, location technologies used to track objects have been analysed. Secondly, architectures to efficiently manage context have been assessed. Thirdly, some interesting examples of context-aware applications have been illustrated. Fourthly, user acceptability of Sentient Computing has been considered. The purpose of this chapter has been to familiarise the reader with context-aware computing and its applications, and highlight areas that still need further investigation. Triggered by these needs, this dissertation describes the implementation of a novel vision-based, fine-grained, 3D location- and orientation-providing sensor. This novel sensor is accessible to everyone and easily deployable everywhere. Moreover, a set of software infrastructure services to aid the implementation of sentient systems is devised. All of these contributions together are deemed to provide the assortment of technologies necessary to make viable a more widespread adoption of the Sentient Computing paradigm in our working and living spaces.

Chapter 3

A Vision-Based Identification Sensor

Most of current personal computers (PCs) feature an IEEE 1394 or USB port that allows the user to capture still images from external video sources. The processing power of PCs is becoming sufficient to enable real-time video image processing. Furthermore, new PCs are often bundled with video cameras in the form of web-cams. Some laptops [SONY01] even come with a built-in swivel camera, which sits in the lid and can deliver video conferencing and general image capture facilities. Surprisingly, very few people have exploited the potential of adding visual awareness to their computers, despite the fact that current off-the-shelf technology permits it.

State-of-the-art video-based sensing is being applied to surveillance and user-authentication applications. Significant progress has been made in the realisation of unmanned CCTV systems [Real01], and in the development of software that can undertake face recognition [Kruizinga01] or iris recognition [Daugman00]. Some companies are currently offering commercial products [Visionics01][eTrue01][Iridian01] that arguably enable rapid and accurate subject authentication or identification within a crowd. Unfortunately, iris recognition requires the subject to be fairly close to the viewing camera and the so-called ‘unmanned’ CCTV systems still require an operator to evaluate the situation when an alarm is triggered by the system. Face Recognition software returns the confidence level with which a subject’s face has been identified. Moreover, it is very difficult to provide generic object recognition software capable of identifying not only people, but also other entities such as computers, pets or books. This is due to the inherent difficulty of the problem of providing visual awareness to computers. This situation has forced researchers to concentrate on a narrower domain, *e.g.* face recognition or iris recognition, whose solution is more viable.

The above arguments have motivated the design of a new vision sensor based on easily-distinguishable artificial landmarks (in the form of tags) attachable to the entities whose identity and 3D location should be determined. Thus, the difficulty of the machine vision task is reduced to the recognition of distinctive markers in an image. This approach, although slightly obtrusive since it requires objects to be tagged, entails much less processing power and permits a more reliable determination of somebody or something's identity.

This chapter illustrates how, using the combination of the processing power of current PCs and conventional low-resolution CCD cameras (CCTV cameras or standard web-cams), a very low additional cost but still effective entity identification and location system can be produced. Firstly, the motivation for undertaking the design of yet another location sensing technology is explained. Secondly, an overview of the novel vision-based sensing technology put forward, termed TRIP, its goals and the design rationale for the visual marker it uses, is given. Thirdly, the image processing techniques required to determine, from a single view of a marker, its identifier and location within an image, are given. Finally, the reliability, frame-processing rate, advantages and limitations of the TRIP sensor are indicated, and some conclusions are drawn.

3.1 The Need for a New Identification/Location Sensor

Since *location* is an important context that changes whenever the user moves, a reliable location-tracking system is critical to many context-aware applications. It is easy to capture such information if the user is willing to cooperate explicitly, *e.g.* by swiping an identification card, pressing a fingerprint reader or logging into a machine. Nevertheless, these methods only provide coarse granularity and low accuracy. Automatic location-sensing is more convenient and usually provides better location accuracy.

Chapter 2 offered an overview of existing location-sensing technologies (recall Table 2.1). Applied to the indoor domain, these technologies extract the identity and location of objects with different degrees of granularity by means of a variety of networked sensors and wireless communication systems. Normally, indoor location technologies were found to be infrastructure-heavy, not cost effective and difficult to deploy in many situations.

Only the CyberCode [Rekimoto+00] and ARToolKit [Kato+99] tagged vision-based systems were found to supply the low cost and easily deployable requirements necessary for a wider acceptance of indoor location technologies in our homes or workspaces. Nevertheless, the design of their markers requires a high camera resolution to correctly extract the object identifier and 3D location of tagged objects from a far distance (*i.e.* more than one metre) from the camera. The coverage range of these technologies is insufficient for their use as indoor object tracking systems, and it explains their intended application for AR.

In the rest of this chapter, a new technology named TRIP is described which offers, in comparison with previous indoor location technologies, a better trade-off between the price, infrastructure complexity, hardware requirements, and the performance and accuracy of the identity and 3D location determined (see last row of Table 2.1).

3.2 TRIP: a Vision-Based Identification/Location Sensor

TRIP[10] (Target Recognition using Image Processing) is a vision-based sensor system that uses a combination of 2D circular barcode or *ringcode* tags (see Figure 3.1), and inexpensive low-resolution CCD cameras to identify and locate tagged objects in a camera's field of view. Relatively low CPU-demanding image processing and computer vision algorithms are applied to obtain, in real-time, the identifier (*TRIPcode*) and pose (location and orientation) of the targets with respect to a viewing camera.

The main features of this new sensor technology are:

1. *Tagged.* It associates a unique, unobtrusive, small passive ringcode tag with each sensed object. This approach differs significantly from other conventional identification or location technologies (with the exception of RFID tags), which

[10] The TRIP system was originally conceived by Alan Jones and Jeremy Henty at AT&T Laboratories Cambridge. They designed the tags, created the original encoding and produced a first prototype of the system. The author took over their unfinished work, chose new, more computationally efficient image processing techniques for the sensor's identification capabilities and devised the pose-extraction method, described in Chapter 4, used to obtain the 3D pose of tagged objects.

usually require electronic battery-powered tags and some complicated wireless technology to transmit their identities to a detector.

2. *Directly interpreted.* TRIP measures the ringcodes' properties of interest, *i.e.* the central bull's-eye's location and identifier, directly without requiring human intervention or the use of complex vision algorithms. This is because TRIP targets are designed to make the recognition process simple, fast and accurate.

3. *Simple infrastructure.* TRIP can be deployed in a standalone fashion with just a camera and a collection of printed targets. Under this application scenario, TRIP can be used for human computer interaction [Moran+99] or AR [Kato+99] [Rekimoto+00]. Alternatively, it could be fed from a number of fixed networked cameras in a building to serve as an indoor location system similar to the Active Badge system.

Figure 3.1. TRIPtag representing the LCE laboratory's front door.

A key design decision for TRIP has been the selection of a suitable marker (a.k.a. target, landmark or fiducial), attachable to objects, which makes their identification and the determination of their 3D location both easy and fast from a computer vision perspective. Thus, only low-cost (below $50), low-resolution (640x480 pixels) web-cams are necessary for TRIP operation. The next section describes the design criteria and layout of the markers.

3.3 A 2D Barcode as a Location Device

1-D barcodes have been used for many years as a cheap machine-readable identifier for a wide variety of objects, e.g. supermarket goods. 2D barcodes [Barcode01] have been proposed more recently for the following three purposes not addressed by 1-D barcodes:

- To allow the barcode to encode some data rather than just an identifier, *e.g.* the sender's and recipient's addresses for a postal parcel.
- To remove the vertical redundancy of the conventional 1-D barcode so enabling the tagging of objects where only a small amount of space is available.
- To represent an easily-distinguishable landmark that can be accurately and easily located using image processing and computer vision techniques.

Similarly to the CyberCode [Rekimoto+00] and ARToolKit [Kato+99] projects, reviewed in section 2.1.6, TRIP attaches a marker to objects in order to facilitate their identification and location. However, rather than concentrating on AR, TRIP aims to serve as a generic indoor location system. TRIP uses 2D ringcodes as *mobile positioning devices.* The design of TRIP's 2D barcode was guided by the following requirements:

- It should be easy to identify both close to and far away from the camera.
- Nothing else can be mistaken for a target.
- It should reliably encode as much information as possible.
- Recognisable using low-resolution CCD cameras and in real-time.

3.3.1 Target Design Criteria

Conventionally, researchers have used visual cues (markers) of known geometric properties to facilitate the identification and localisation of objects in the field of view of video cameras. Markers of known size and shape can yield a wealth of geometric information when transformed under the perspective projection. In particular, circular [Thomas+97] and square [Kato+99][Rekimoto+00] markers have been utilised to extract the accurate position and orientation of tagged objects. These markers permit the application of simple and fast-performing image processing and computer vision algorithms to the difficult task of obtaining 3D position from a 2D image.

The TRIP marker (see Figure 3.1) is based on a circular pattern since circles are more salient than squares for their identification and 3D localisation using a visual sensor. In man-made environments, circles are less common shapes than right angles, squares and rectangles. TRIP is intended for the localisation of tagged objects in these highly cluttered environments (see Figure 3.2), where the bull's-eye of a TRIPtag represents a very distinctive pattern. The detection of squares in those cluttered environment supposes an expensive computational task given the many straight edges combinations possible within an image. The argument that unusual shapes are more easily distinguishable could have been taken to an extreme by choosing an even less common shape (*e.g.* a star). In this case, however, there is the problem of representing such a shape in a compact way and to encode efficiently information (*e.g.* ID) around or within it. A circle, and its projection in an image as an ellipse, can be represented in a compact and elegant form by a symmetric 3x3 matrix (see equation (3.7)). In addition, ellipses in an image are salient features and can be detected robustly and rapidly by methods such as [Pilu+96].

Figure 3.2. View of a 7x7 cm TRIP marker in a cluttered environment.

A second important aspect in the design of a marker, apart from its shape, is the set of colours used in order to provide a distinctive contrast between the marker and the surrounding objects. TRIP targets were chosen to be black and white patterns since: (1) it is computationally cheaper to process monochromatic images rather than colour ones, and (2) monochromatic printers are also more widely available than colour ones. Using colour is both difficult and unreliable since the lighting conditions can change too much and the colour sensitivity of various CCD devices are very different. In addition, the memory and CPU needs for the parsing of greyscale images are lower by a factor of three than in the

case of colour images. Note that a pixel in RGB24 format occupies three bytes of storage whereas a greyscale pixel only requires a single byte.

3.3.2 The TRIP marker: TRIPtag

A TRIP marker, or *TRIPtag*, (see Figure 3.3), encoding a ternary identifier, presents the following main features:

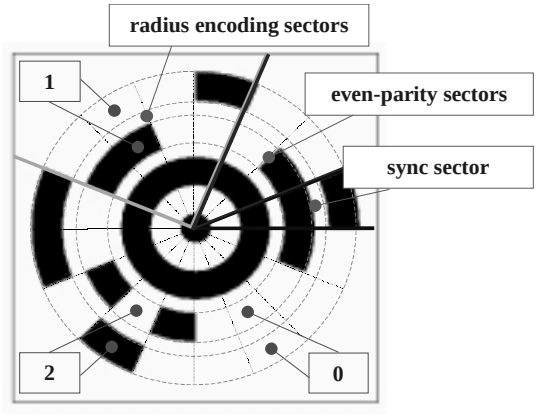


Figure 3.3. TRIPcode of radius 58mm and ID 18,795.

- A central *bull's-eye* makes the identification process easier due to its properties of invariance to rotation and perspective, and high contrast.
- Two code rings around the bull's-eye encode its ternary identifier. Each ring code provides 16 bits of information that are read in anti-clockwise fashion. Observe that to encode a '0' ternary digit two blank areas are left within a sector. To encode a '1', the area corresponding to the innermost encoding ring is drawn black and the area corresponding to the second encoding ring is left white. Finally, to encode a '2' ternary digit the reverse approach to the case of '1' is applied. The 16 sectors of a ringcode are utilised in the following form:
  - The first (or *synchronisation)* sector's special and elsewhere impossible configuration, *i.e.* two black sections in the encoding rings' areas, indicates the beginning of the code. This factor prevents the use of quaternary codes, which could give place to a larger address space.
  - The second and third sectors are used to implement an even parity error check.
  - The four subsequent sectors, in anti-clockwise fashion, encode (in ternary) the radius of the outermost border of the bull's-eye.

- The remaining nine sectors[11] encode an identifier or *TRIPcode* in ternary. Therefore, the maximum number of valid codes is $3^9 = 19{,}683$.

The dimensions of the tag, along with the resolution of the camera, determine the maximum distance from the camera at which the tag may be identified. The high contrast, rotationally invariant design of the tag enables a high identification success rate even when the tag occupies only a small number of pixels in the captured image.

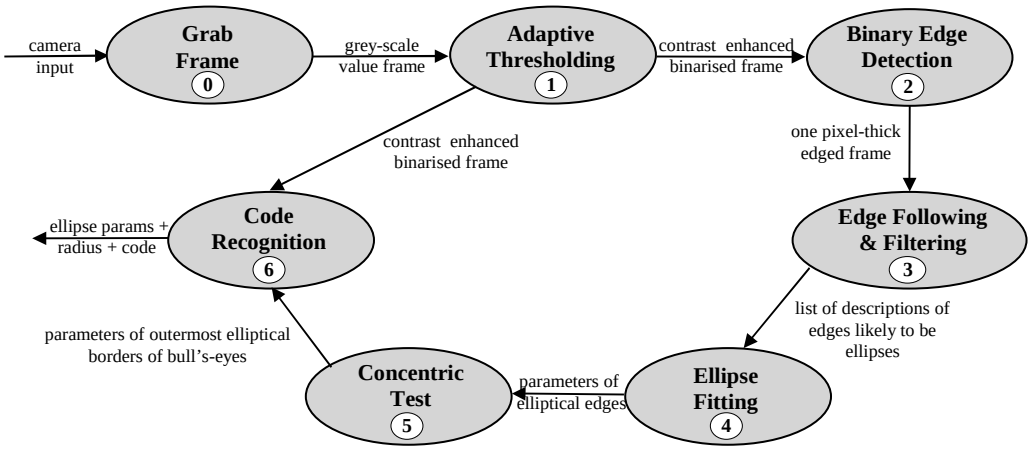


Figure 3.4. Target Recognition Process stage pipeline.

3.4 Target Recognition Process

With exception of the camera and tags required, TRIP is an entirely software-based identification and location sensor. TRIP applies a target recognition algorithm to video data supplied by a camera in order to determine the identifier and geometric properties of the projections of TRIPtags in a frame. The discovered geometric properties of TRIPtag projections are used by a separate method to determine the 3D location and orientation,

[11] In the original design of the TRIPcode, the four sectors reserved for encoding the radius of the outermost border of a target's bull's-eye's were also used for identifier encoding. Therefore, the number of valid TRIPcodes was $3^{13} = 1{,}594{,}323$. The radius encoding was introduced since knowledge of the radius of a target's bull's-eye is a necessary requirement for the extraction of a target's pose, given an image of it. Figure 4.14, Figure 5.6, Figure 8.2, Figure 8.4 and Figure 8.5 depict TRIPcodes based on the original design.

with regard to the viewing camera, of TRIP-tagged objects. Chapter 4 describes this pose extraction method. This section concentrates on the description of the target recognition process, *i.e.* a set of image processing stages that filter out irrelevant information from the originally grabbed image in order to extract the identification of TRIP targets. Figure 3.4 shows schematically the video filtering process.

3.4.1 Image Acquisition (Stage 0)

Frame Grabber software[12] is used to acquire greyscale digital images from a video camera attached to a computer via either a TV video card, a USB port or an IEEE 1394 (*firewire*) port. A digital image is represented by a numerical matrix, I, with n rows and m columns, where $p_n=[x,y]$ denotes the image value (brightness) at pixel (x, y) (x-th column and y-th row), and encodes the intensity recorded by the photo-sensors of the CCD array contributing to that pixel. The TRIP system makes exclusive use of monochromatic images, where p_n represents a greyscale value in the range 0 (black) to 255 (white).

3.4.2 Adaptive Thresholding (Stage 1)

Thresholding [Davies97] [González+93] is an image segmentation technique that aims to separate objects from the background. This process removes the effects of uneven lighting, glints and shadowing in the field of view and increases the overall contrast of the image. In TRIP, a thresholding stage is applied to the raw greyscale images captured for the following reasons: (1) the video frame sources may correspond to low-quality, low-cost, security cameras or web-cams with low resolution and contrast level, and high noise, and (2) to compensate for varying and uncontrolled lighting conditions. The remaining of this section discusses two different forms of thresholding and argues why the second form was chosen.

[12] Two of the most common video capture APIs are the video4linux API (http://www.exploits.org/v4l/) and Microsoft Research's Vision SDK (http://research.microsoft.com/projects/VisSDK/). Several open source frame grabbers exist that make use of these APIs.

3.4.2.1 Global Thresholding

The simplest way of thresholding is to set all pixels below a certain grey level to black (0) and clear all others to white (255). This process is called *global thresholding* because every pixel, p_n, in the image is compared against the same fixed threshold value, T:

$$globalThres(n) = \begin{cases} 0 & if\,(p_n < T) \\ 255 & otherwise \end{cases} \quad \textbf{(3.1)}$$

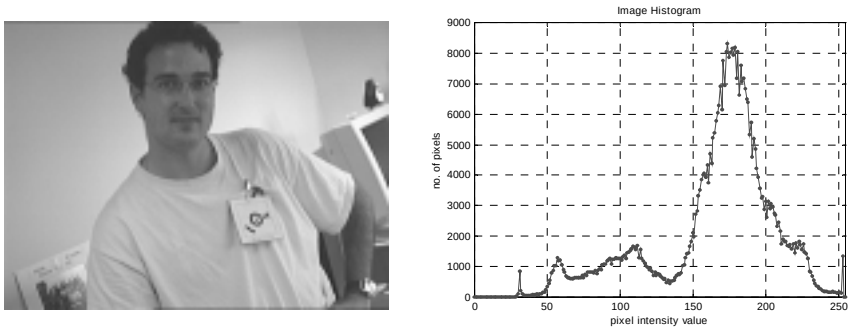


Figure 3.5. Histogram of image.

The key problem here is how to select a suitable threshold value, T. One possibility is to pick the centre of possible values, so if pixels are eight bits deep (ranging from 0 to 255), then 128 would be selected. This approach works well if all the 'dark' pixels of the image really do have values under 128, and light pixels have values above 128. But if the image is over or under exposed to light, this criteria does not work. In fact, it is better to look at the range of *actual* values instead of *possible* values to determine the threshold. A common method used to select the threshold is not just to look at the range of actual values, but also at their distribution. A histogram of the pixel intensities should, theoretically, show a large background peak, as well as a smaller peak corresponding to dark objects. The whole curve may be shifted to the left or right depending on the ambient light level, but in any case, the best value to pick for thresholding is the local minimum between the two peaks. Unfortunately, due to image noise and uneven lighting, those peaks are often not easily distinguishable as shown by Figure 3.5, and more than two peaks are identified.

The global thresholding approach assumes that all the areas captured by an image are evenly lit. Hence, problems arise when illumination is not sufficiently uniform. Moreover, it involves two passes through the image, one to calculate a suitable threshold value and the second one to apply it to the image. In principle, the TRIP system could select a threshold value periodically and later apply the same value for all the images processed in each given period. However, this approach still assumes that the same threshold value is applicable to the whole image, not taking into consideration the effects of glints, shadows and clutter within a region of an image. Those inconveniences led to the search for a thresholding method that would adaptively vary the threshold value across the image and would still only require a single pass through the image. The *adaptive thresholding* algorithm described in [Wellner93] was chosen because it converts greyscale images into binary images reliably at a low computational cost.

3.4.2.2 Wellner's Adaptive Thresholding

Wellner's adaptive thresholding method varies the threshold value used to transform a pixel into black or white value based on the background illumination of that pixel. The basic idea is to run through the image while calculating a moving average of the last *s* pixels seen. When the value of a pixel is significantly lower than this average, then it is set to black, otherwise it is set to white. Only one pass through the image is necessary. Let p_n represent the value of a pixel at point *n* in the image. Let $f_s(n)$ be the sum of the values of the last *s* pixels at point n:

$$f_s(n) = \sum_{i=0}^{s-1} p_{n-i} \quad \textbf{(3.2)}$$

The value of the resulting image *T(n)* is either 0 (for black) or 255 (for white) depending on whether it is *t* percent darker than the average value of the previous *s* pixels. In the implementation of TRIP, values of $1/8^{th}$ of the width of the image for *s* and of 15 for *t* were established experimentally. These values yielded very good results in an office scenario where shadowing effects and moderately varying light conditions are common.

$$T(n) = \begin{cases} 0 & if \quad p_n < \left(\frac{f_s(n)}{s}\right) \cdot \left(\frac{100-t}{100}\right) \\ 255 & otherwise \end{cases} \quad \textbf{(3.3)}$$

A faster way to calculate an approximate (weighted) moving average is to subtract *1/s* part of it and add the value of only the latest pixel instead of using all *s* pixels. Thus, *g(n)* represents a more computationally efficient approximation of *f(n)*:

$$g_s(n) = g_s(n-1) - \frac{g_s(n-1)}{s} + p_n \quad \textbf{(3.4)}$$

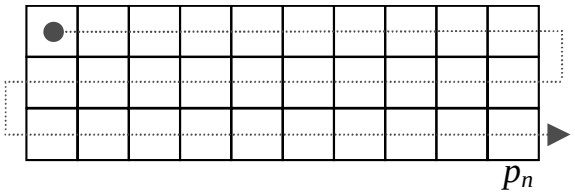


Figure 3.6. Traversing pixels in alternate direction every other line.

In order to make the algorithm work equally well irrespective of whether the light source is on the left or right hand side of the camera, Wellner's method transverses the image alternatively from the left and from the right (see Figure 3.6). However, in grey regions of an unevenly lit image, going in one direction can produce the opposite result to going in the other, thereby producing an every-other-line effect. In order to solve this problem, Wellner suggests keeping the previous line of approximate averages (which were calculated by scanning in the opposite direction) and calculating the media between the current line-average with the previous line's average. Thus, *h(n)* instead of *g(n)* can be used:

$$h(n) = \frac{g(n) + g(n - width)}{2} \quad \textbf{(3.5)}$$

a) Global Thresholding **b) Adaptive Thresholding**

Figure 3.7. Global vs. Adaptive Thresholding.

Figure 3.7 shows the results of applying to the image in Figure 3.5, a global threshold of value 141 and Wellner's adaptive algorithm, respectively. Observe that Wellner's method operates better enhancing the contrast of the pixels belonging to the TRIPtag.

3.4.3 Binary Edge Detection (Stage 2)

Edge detection is one of the most commonly used operations in image analysis and there are probably more algorithms in the image processing literature [González+93] [Sonka+99][Parker97] for enhancing and detecting edges than for any other single subject. The reason for this is that edges form the outline of an object.

Edges correspond to the pixels at or around which the image values undergo a sharp variation. Conventionally, edge detection techniques apply the geometric interpretation of the *gradient* (the measure of change in a function) to an image, expressed as the rate of change of the grey levels in it. This rate of change is large near an edge and small in constant areas. The Canny edge detector [Canny86] is probably the most commonly used in today's machine vision community.

For TRIP, given that the output of the Adaptive Thresholding stage is a binary image with only black and white pixel intensity values, it is not necessary to apply a fully-fledge edge detection method. The binary edge detector presented is not as precise as other morphological or gradient-based edge detection methods, only offering pixel (rather than sub-pixel) accuracy in the edge location. However, the difficulty of other more

sophisticated edge detectors is high and requires much more computational power. Still, sub-pixel accuracy on the concentric borders of a target is partially recovered by the ellipse detection stage shown in section 3.4.5.

The process of finding the edges of a binary object may be carried out in different forms. Clearly, it can be assumed that an edge point is one that is not completely surrounded by only black pixel values or only white pixel values. However, it may be defined as being within the object, within the background or in either position. Taking the assumption that the edge of an object has to lie within the object, the following edge-finding heuristic for binary images can be formulated:

"A pixel in a binary image is an edge point if it has black intensity value and a 4-connected neighbour pixel with white intensity value and an 8-connected one with black intensity value"

Figure 3.8 illustrates what it is meant by 4-connected or 8-connected neighbour of a pixel. It also shows the numbering scheme used to name the 8-connected neighbouring pixels of a central pixel, denoted by p_n. This numbering scheme serves to calculate two essential parameters required to undertake the binary edge detection of an image, namely *sigma* and *gamma*. The *sigma* value of pixel p_n is defined as the sum of the pixel values surrounding it, *i.e.* `sigma=P0+P1+P2+P3+P4+P5+P6+P7`. The *gamma* value of pixel p_n is the sum of the pixel values corresponding to its diagonally adjacent neighbour pixels, *i.e.* `gamma = P1+P3+P5+P7.`

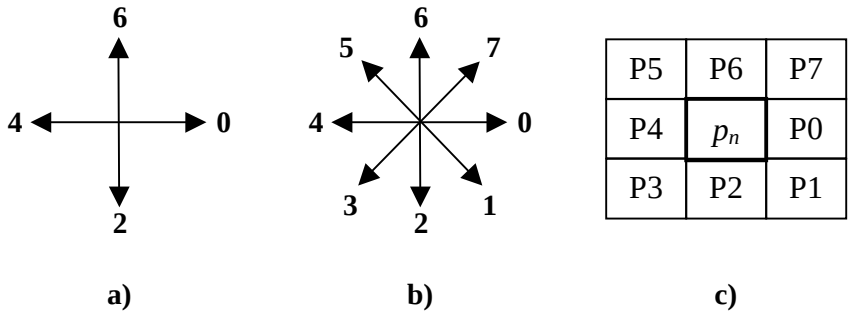


Figure 3.8. Pixel neighbourhood notation: a) 4-connectivity; b) 8-connectivity; c) 8-connected neighbours of a central pixel denoted by p_n.

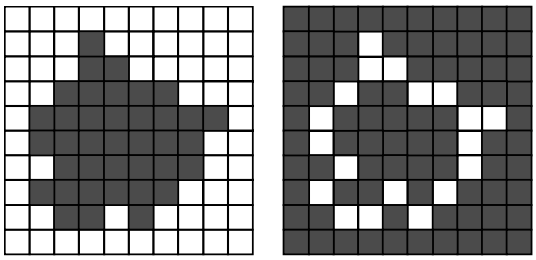


Figure 3.9. Result of applying binary edge detection.

The strategy applied to identify the edges of a black region within a binary image amounts to cancelling out object pixels that are not on the edge. Moreover, those pixels that correspond to an edge but are not 4-connected to any neighbouring background (white) pixels are also eliminated. Figure 3.9 shows the result of applying the edge detection algorithm outlined in Figure 3.10 to a binary image. This algorithm produces one-pixel thick edges in a very CPU efficient manner with pixel-accuracy. Sub-pixel accuracy can be obtained using other more computationally intensive gradient-based edge detection algorithms. However, a main goal in the design of TRIP is to achieve real-time performance with standard off-the-shelf PCs. This is the reason why the more computationally efficient, but less accurate, binary edge detection algorithm was chosen.

```
FOR EACH pixel p_n ∈ THRESHOLDED_IM AND e_n ∈ EDGED_IM:
  IF p_n = BLACK:
    sigma = P0+P1+P2+P3+P4+P5+P6+P7
    gamma = P1+P3+P5+P7
    IF ((gamma>0) AND (gamma=sigma)) OR (sigma=0):
      e_n = BLACK // no edge point
    ELSE:
      e_n = WHITE // edge point
  ELSE:
    e_n = BLACK // no edge point
```

Figure 3.10. Algorithm for edge detection in a binary image.

3.4.4 Edge Following and Filtering (Stage 3)

In order to fit the boundary of a pattern to a model, *e.g.* an ellipse, is necessary to store an ordered list of edge points (*edgel*) locations in a data structure. Thus, all the 8-connected chains of edgels previously located are followed in a clockwise fashion, producing for each edge a list of ordered point locations. Figure 3.11 illustrates how the previously calculated edge can be tracked. Starting from the first edgel encountered while scanning the image from top to bottom and left to right, this method searches for the next edgel to

the current one in the order shown by the left hand side of Figure 3.11. Note that every edge point that is assigned to an edge must be made black in order to prevent the method from looping through already visited edge points.

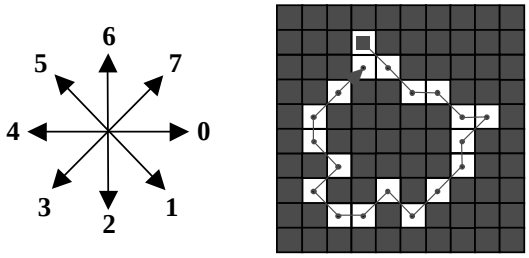


Figure 3.11. Edge Tracking of 1 pixel thick edge using 8-connectivity.

The circular bull's-eyes of TRIPtags are observed in the captured frame as elliptical due to perspective projection. In fact, TRIP is looking for the identification of at least two 'concentric'[13] edge ellipses that may correspond to a candidate target's bull's-eye. This is the reason why this stage applies a filtering process to every edge tracked, retaining only the ones plausibly belonging to an ellipse. The criteria undertaken filters out edgel chains whose ratio between its perimeter and the distance between the initial and final points is bigger than an empirical value greater than 1. This ratio equals 1 in the case of straight lines. Experimentally it has been established that a value of this ratio of 5 maintains the edges likely to define elliptical arcs.

3.4.5 Ellipse Detection (Stage 4)

The previous stage provides edges likely to define an elliptical shape. In this stage, an ellipse detection [Fitzgibbon+95] method is applied which seeks for each of these edges a conic function representing an ellipse, given by the following implicit equation:

$$f(p,a) = x^T \cdot a = ax^2 + bxy + cy^2 + dx + ey + f = 0 \quad \textbf{(3.6)}$$

, where $p = [x \quad y]^T$ is an image point, $a = [a \quad b \quad c \quad d \quad e \quad f]$ and $x = [x^2 \quad xy \quad y^2 \quad x \quad y \quad 1]^T$. This equation can be represented in matrix form as:

[13] Note that where it is said concentric, it is really meant 'approximately concentric', due to the fact that spatial distortions make the ellipses not to be concentric in the pure mathematical sense.

$$X^T \cdot C \cdot X = 0 \text{, where:}$$

$$C = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \text{ and } X = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \textbf{(3.7)}$$

The solution sought is the choice of parameters of an ellipse that best matches the observed locations of the given points in the edge in the image, $p_i=[x,y]^T$, in a *least squares sense*. This can be achieved by minimising the Euclidean distance between points belonging to the ellipse, p, and the measured points, p_i, in the image. This is:

$$\min_a \sum_{i=1}^{N} \left\| p - p_i \right\|^2 \quad \textbf{(3.8)}$$

where a stands for the parameter vector associated to the ellipse which fits the image points p_i best, in the least squares sense, and p belongs to the ellipse:

$$f(p,a) = 0$$

Pilu et al. [Pilu+96] show that if instead of the Euclidean distance, the algebraic distance is used, a closed form solution can be obtained. The problem then becomes:

$$\min_a \sum_{i=1}^{N} \left| x_i^T \cdot a \right|^2 \quad \textbf{(3.9)}$$

where the factor $x_i \cdot a$ represents the algebraic distance of a image point p_i from a curve $f(p,a)=0$, *i.e.* $\left| f(p_i, a) \right|$ (see equation (3.6)). This method, chosen for TRIP, provides an ellipse-specific solution with low processing demands. Moreover, it finds the elliptical borders of a TRIPtag with sub-pixel accuracy, recovering part of the accuracy lost experienced after applying binary edge detection.

The six parameters defining a conic representing an ellipse returned by this method are transformed into the five parameters that define an ellipse in a 2D coordinate system (see Figure 3.12). This representation is more convenient for the two following stages of the target recognition process. [Farin+97] provides a complete description of how to recover the five parameters of an ellipse from its implicit equation.

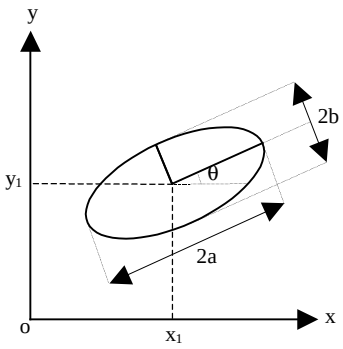


Figure 3.12. Ellipse with centre at (`x1, y1`), width `a`, height `b`, rotated by `θ`.

3.4.6 Concentric Test (Stage 5)

The ellipse parameters obtained in the previous stage are compared to identify concentric ellipses likely to form part of the projection of candidate targets' bull's-eyes. Potentially three concentric ellipses should be identified per TRIPtag's central bull's-eye view. However, the innermost circle is often too small and its pixel values are filtered out by the previous image processing stages. Still, in order to avoid spurious candidates of TRIPtag sightings, the intensity value of the pixel at the centre of the concentric ellipses in the outcome of the thresholding stage is tested to check whether it corresponds to a black intensity value. This is the reason why the innermost black circle in a target is necessary.

3.4.7 Code Deciphering (Stage 6)

A code extraction stage is applied to the candidate TRIP sightings. This method applies an efficient pixel-sampling mechanism on the binary image result of the Adaptive Thresholding stage, based on the parameters of the outermost ellipse of the projection of a candidate TRIPcode. The pixel-sampling mechanism requires the following two steps:

1. *Identify the synchronisation sector*. It is necessary to locate the beginning of a TRIPcode. For that purpose, the bull's-eye ellipse of reference is transformed to the unit circle, since the ratios between the radius of the bull's-eye and the code rings circular borders are only known with respect to the TRIP target design. Figure 3.13 shows in the right hand side the dimensions of the sector highlighted in the left hand side. The measurements depicted represent the radii of each circular border within a TRIPtag. The transformation of an ellipse point $p = [x\ y]^T$ to one in the unit circle is given by:
    a. The translation of the reference ellipse centre to the origin:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -x_1 \\ -y_1 \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \quad \textbf{(3.10)}$$

    b. Rotating the ellipse by –θ:

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \end{bmatrix} \quad \textbf{(3.11)}$$

    c. Compressing the axes:

$$\begin{bmatrix} x''' \\ y''' \end{bmatrix} = \begin{bmatrix} x''/a \\ y''/b \end{bmatrix} \quad \textbf{(3.12)}$$

    Once this is done, the intersection points between an arbitrary line passing through the centre of the unit circle, and the two imaginary circumferences going through the middle of the ring codes are determined. These intersection points are transformed back to the corresponding image location using the inverse transformation to the one employed to convert the reference ellipse point to a unit circle point. If the intensity values of these points, sampled on the output of the Adaptive Thresholding stage, simultaneously correspond to the black intensity value in the two ring codes, then the synchronisation sector has been identified. This process is repeated iteratively by rotating the reference line 15° and calculating the new intersection points. If the synchronisation sector cannot be found after 12 iterations, the candidate bull's-eye is spurious and rejected. If found, the right most border of the synchronisation sector is located by sampling points in clockwise sense until two black points are no longer found within the sector. Then, the bottom outermost corner of the synchronisation sector is found by pixel sampling along the last position of the reference line. This point is of importance for the pose extraction method described in chapter 4.

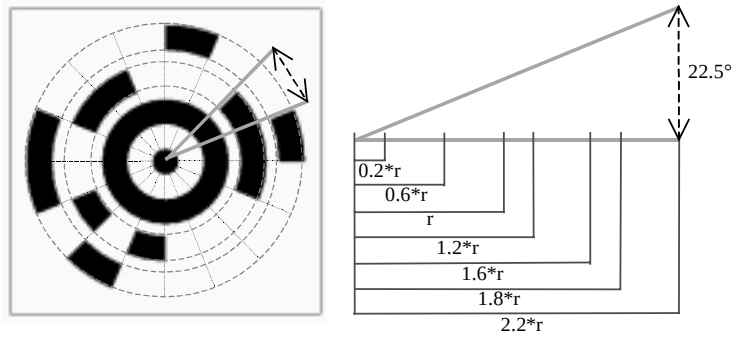


Figure 3.13. Dimensions of a TRIP target.

2. *Decode the barcode identifier*. After the previous operation is completed, sample points in the middle of each of the 22.5° code sectors are taken following the same transformations. If the black colour intensity value is found in a point belonging to the first ring code, the ternary value in that sector is 1, if in the second is the value 2, and otherwise is a 0. The ternary codes obtained are validated against the two even parity check bits reserved in a TRIPcode (see Figure 3.3).

3.4.8 Target Recognition Process Outcome

The outcome of the target recognition process is a list of TRIP sighting descriptions. Each TRIP sighting contains: (1) a target's ternary identifier, (2) the radius of its central bull's-eye's outermost circular border, (3) the geometric properties of this border's elliptical projection in the image (x_1, y_1, a b, and θ), (4) the six parameters of the conic corresponding to the bull's-eye's outermost circular border's elliptical projection and (5) the location in the image of the bottom outermost corner of the synchronisation sector of a TRIPtag.

Figure 3.14 demonstrates the target recognition process over an example frame. The outcome of the intermediary stages (when visible) is shown. On the last image, a crosshair marks the centre of the target identified. In addition, the results of the recognition process are displayed together with the results provided by the Pose Extraction Algorithm, *i.e.* the target's 3D location and orientation with regards to the viewing camera. This method is discussed in chapter 4.

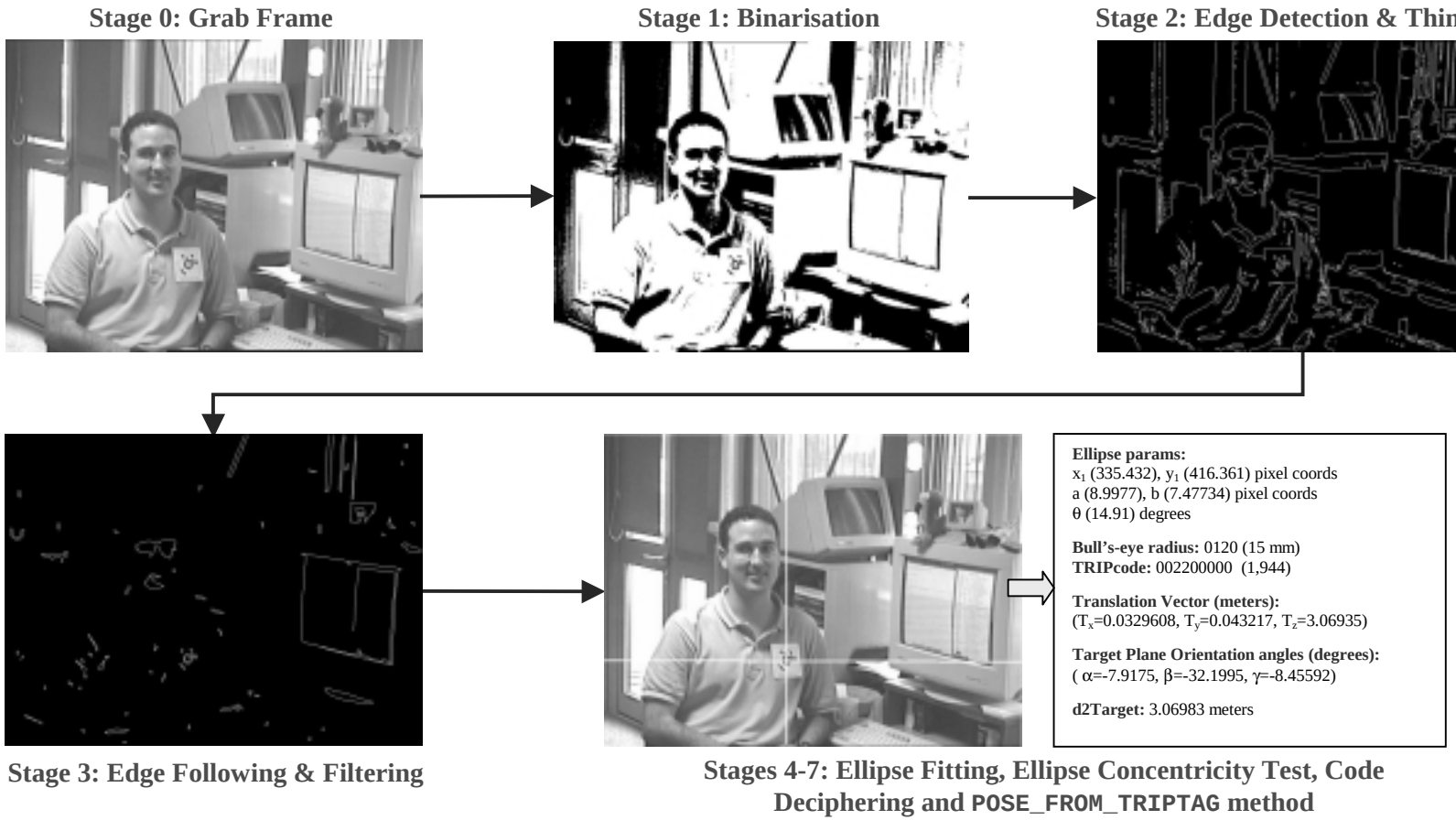


Figure 3.14. Identifying and Locating a TRIPtag in a 768x576 image.

3.5 TRIP Sensor System Evaluation

This section briefly summarises the reliability and performance of a C++ implementation for the target recognition process. In addition, it assesses the principal limitations of the TRIP visual sensing technology.

3.5.1 TRIP Reliability and Performance

A thorough discussion of TRIP results is postponed until the next chapter (section 4.5) where the full capabilities of the visual sensing system suggested are completely covered. At that point, the identification capabilities of TRIP will be complemented with 3D location capabilities. The C++ implementation of the target recognition process is capable of running at a frequency of 17 Hz on a 1.4 GHz AMD Athlon machine, when parsing frames of 640x480 pixels in size. Under normal lighting conditions, a successful target sighting occurs when the image of a target occupies an area of at least 40x40 pixels and the slant between the target plane and the camera is below 70 degrees. If those threshold conditions are not met the system may be unable to locate the targets. Moreover, if the target is very close to the camera (closer than 10 cm) the borders of a TRIPtag will not appear elliptical and the recognition process may fail.

3.5.2 TRIP Limitations

The main limitation of TRIP is its *line-of-sight* requirement between a target and a camera. Several cameras are required to ensure comprehensive tracking coverage of an indoor environment due to TRIP's sensitivity to occlusions. Partial occlusions where the central bull's-eye can be clearly seen but the identifier encoding rings are partially blocked may be overcome by using redundant code bits. A second inconvenience of the TRIP sensor system is that it requires environments that are well illuminated, *e.g.* a conventional office environment. It is impossible to locate a target in dark conditions unless the TRIPtag is printed in a fluorescent material or infrared light is used. Finally, the passive nature of TRIPtags prevents them from being used as simple input to control devices, in the way used by other technologies such as the Active Badge or Bat systems.

3.6 Conclusion

This chapter has described a new vision-based identification sensor, named TRIP. The main advantages of TRIP are its low cost and easy deployability. TRIP only requires off-the-shelf hardware, *i.e.* the processing power of a conventional PC, a low-resolution CCD camera to capture images (*e.g.* a web-cam that costs less than $50), and a printer to produce the specially designed 2D ringcodes. Compared with other existing indoor identification and location systems, TRIP can be installed without requiring expert knowledge or involving the purchase and deployment of complicated hardware and networking technologies. Moreover, TRIP's unique marker design makes this sensor technology inexpensive and versatile. The low-cost and reasonably sized address space of TRIP markers (19,683 IDs) make this system suitable for tagging even low-cost items, such as books or stationery. In contrast, other location sensing technologies can only tag valuable assets, such as personnel or equipment. TRIP's software nature enables users with a web-cam attached to their PCs to quickly install the TRIP software and hence provide visual awareness to their machines. Chapter 4 extends the TRIP sensor with 3D location capabilities.

Chapter 4

An Adaptive Vision Location Sensor

Chapter 3 described the design of a marker-based object identification sensor, namely TRIP. It showed that by tagging objects with specially designed tags, the image processing required for object identification is simplified enormously compared to the processing demands of untagged vision-based systems. This chapter discusses a method for extracting the location and orientation, *i.e. pose*, of TRIP-tagged objects. Given a single 2D image of a tagged object, the method recovers the object's 3D location and orientation with respect to the viewing camera. Based on the convenient geometric properties of the TRIP marker, this method efficiently and accurately extracts the pose of objects. The resulting sensor is both an identification and 3D location sensor, operating in real-time and requiring only off-the-shelf hardware.

This chapter describes the application of a *model-based object location* method for finding the pose of a planar TRIPtag in space. Firstly, some essential concepts on projective geometry are introduced in order to prepare the reader for the mathematics of the method. Secondly, a `POSE_FROM_TRIPTAG` method to extract the position and orientation of tagged objects is presented. Thirdly, an adaptive algorithm devised to alleviate the relatively high processing demands of the TRIP system is outlined. The chapter concludes by showing the accuracy and performance measurements of the TRIP location sensor.

4.1 Projective Geometry

This section introduces some fundamentals of projective geometry. No attempt is made to provide a comprehensive survey of this subject, which can be found in many good works

on computer vision, such as [Faugeras93], [Trucco+98], [Hartley+00] or [Pollefeys00]. Readers familiar with Computer Vision may skip the following explanations and proceed to section 4.3.

Projective Geometry, when applied to computer vision, studies the use of 2D image information for automated measurement of the 3D world. One application of projective geometry is object location. The aim of *object location* is to link the position of 3D scene points with that of their corresponding 2D image points. For this, the geometric projection performed by the vision sensor (or *perspective projection*) needs to be modelled. The pinhole camera model is conventionally chosen to depict this transformation.

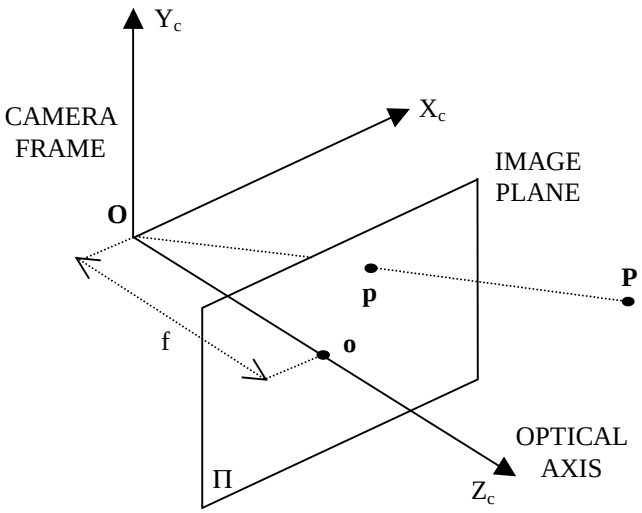


Figure 4.1. Pinhole camera model.

4.1.1 Pinhole Camera Model

The most common mathematical model of an intensity camera is the perspective or *pinhole camera model*. The geometry of this model is depicted in Figure 4.1. It consists of a plane Π, the *image plane*, and a 3D point *O*, the *focus* or *centre of projection*. The distance between Π and *O* is the *focal length* (*f*). The line through *O* and perpendicular to Π is the *optical axis*, and *o*, the intersection between Π and the optical axis, is the *image centre* or *principal point*. Based on this model, a camera undertakes a perspective

projection from the 3D projective space P^3 to the 2D projective plane P^2. This projection is performed by an optical ray (also known as a light beam) reflected from a scene point *P*. The optical ray passes through the centre of projection, *O*, and the image plane, Π, at the point *p*.

Consider the 3D reference frame in which *O* is the origin and the plane Π is orthogonal to the *Z*-axis, as shown in Figure 4.1. Let $P=[X_c, Y_c, Z_c]^T$ and $p=[x_c,y_c]^T$ (where *T* indicates transpose), represent a 3D point in space and its 2D image, respectively. This reference frame, called the *camera frame* or *camera coordinate system*, can be used to represent the two fundamental equations of perspective projection, shown in equation (4.1).

Fundamental Equations of Perspective Projection

In the *pinhole camera model* (non-linear), the coordinates $[x_c,y_c]$[14] of a point *p*, image of the 3D point $P=[X_c,Y_c,Z_c]^T$, are given by:

$$x_c = f \cdot \frac{X_c}{Z_c}$$

$$y_c = f \cdot \frac{Y_c}{Z_c}$$

(4.1)

Homogeneous coordinates play a role in projective geometry equivalent to that which Cartesian coordinates play in Euclidean geometry. They offer a more natural framework for the study of projective geometry. Through them, the imaging process can be expressed as a *linear* matrix operation in homogenous coordinates. Furthermore, a series of projections can be expressed as a multiplication of matrices. These features are extensively used in this chapter.

Consider a point X in the *n*-dimensional space with Cartesian coordinates given by the *n*-tuple $(X_1, X_2, \ldots, X_n) \in \Re^n$. The expression of X in homogenous coordinates is the set of (n+1) tuples $\{w(X_1, X_2, \ldots, X_n, 1) \; \forall \; w \in \Re^n-\{0\}\}$. Conversely, given the homogenous coordinates $\{w(X_1, X_2, \ldots, X_n, X_{n+1}) \in \Re^{n+1}-\{0, 0, \ldots, 0, 0\}, \; \forall \; w\neq 0\}$ of a point X in the n-dimensional space, the Cartesian coordinates of X will be given by $(X_1/X_{n+1}, X_2/X_{n+1}, \ldots, X_n/X_{n+1})$, if $X_{n+1}\neq 0$. If $X_{n+1}=0$, the point X is said to be at infinity in direction $(X_1, X_2, \ldots, X_n)$, and it cannot be represented in Cartesian coordinates.

Figure 4.2. Homogenous coordinates sidebox.

[14] Note that the third component of an image point is always equal to the focal length (as the equation of the plane Π is $z_c=f$). Therefore, it is written $p=[x_c,y_c]^T$ instead of $p=[x_c,y_c,f]^T$.

The expressions in equation (4.1) are *non-linear* because of the factor $1/Z_c$, and do not preserve either distances between points, or angles between lines. They can be made *linear*, however, by using *homogenous coordinates*. A brief overview of homogenous coordinates is given in Figure 4.2.

Linear Version of Fundamental Equation of Perspective Projection

In the *pinhole camera model*, the homogenous coordinates $[x_c', y_c', s]$ of a point p, image of a 3D point P, expressed in homogenous coordinates as $[X_c, Y_c, Z_c, 1]^T$, are given by:

$$\begin{bmatrix} x_c' \\ y_c' \\ s \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad \textbf{(4.2)}$$

Here $x_c = x_c'/s$ and $y_c = y_c'/s$, represent the Cartesian coordinates of the point p in the image plane. The factor s denotes an arbitrary scale factor.

4.1.2 Camera Parameters

Machine vision algorithms that compute the position of objects in space need equations linking the coordinates of points in 3D space with the coordinates of their corresponding 2D image points. Unfortunately, equation (4.2) is not useful in practise as the position and orientation of the camera frame is unknown and not directly accessible. However, it is often assumed that:

- The camera reference frame can be located with respect to some other, known, reference frame (the *world reference frame*).
- The coordinates of the image points in the camera reference frame can be obtained from *pixel coordinates*, the only ones directly available from the image.

These assumptions are equivalent to having knowledge of some camera's characteristics, *i.e.* the camera's *extrinsic* and *intrinsic* parameters. The process of estimating the value of these parameters is called *camera calibration*.

4.1.2.1 Intrinsic Parameters

The *intrinsic parameters* are defined as the set of parameters needed to characterize the optical, geometric, and digital characteristics of the viewing camera. For a pinhole camera, three sets of intrinsic parameters are needed:

- The perspective projection, for which the only parameter is the focal length *f*.
- The transformation between camera frame coordinates and pixel coordinates.
- The geometric distortion introduced by the optics.

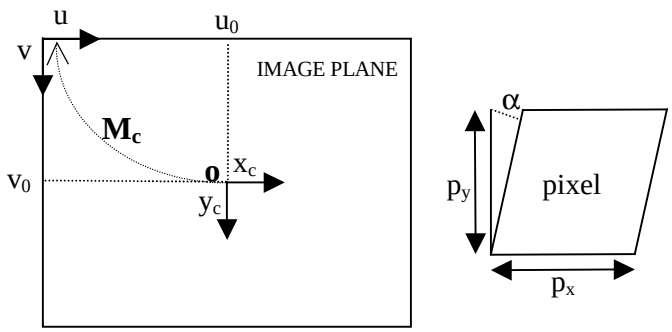


Figure 4.3. From image reference frame coordinates to pixel coordinates.

The pixel coordinates in the image do not correspond to the physical coordinates in the image plane. With a CCD camera the relation between both depends on the size and shape of the pixels and on the position of the CCD chip in the camera. To find the transformation (M_c) between the camera and pixel coordinates, the coordinates (u,v) of an image point in pixel units must be linked with the coordinates (x_c,y_c) of the same point in the camera reference frame as shown in Figure 4.3. Moreover, there may be skew (non-orthogonality) between the pixel axes, α, which for standard cameras is represented by a factor k_α with a value close to zero.

The transformation between Camera and Image Frame Coordinates

To model CCD imaging, a mapping M_c, between the pixel coordinates $w=(u',v',s)$ and the image plane coordinates $p=(x_c',y_c',s)$, in homogenous form, is defined:

$$\begin{bmatrix} u' \\ v' \\ s \end{bmatrix} = M_c \cdot \begin{bmatrix} x_c' \\ y_c' \\ s \end{bmatrix}, \text{ where } M_c = \begin{bmatrix} k_u & k_\alpha & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(4.3)}$$

Here, the Cartesian pixel coordinates (u,v) are obtained by the factors u'/s and v'/s respectively.

Therefore, the intrinsic parameters of a camera are defined as the focal length, f, the location of the image centre in pixel coordinates (u_0, v_0), the effective pixel size (in mm) in the horizontal and vertical direction (k_u, k_v), and the skew k_α. For most CCD cameras the pixels are almost perfectly rectangular and thus $k_\alpha \approx 0$. For a camera with fixed optics these parameters are identical for all the images within the camera. For cameras with zooming and focusing capabilities the focal length and the principal point location in the image can vary. Nevertheless, for auto-focusing cameras the intrinsic parameters vary only slightly and can reasonably be considered fixed. For auto-zooming cameras the intrinsic parameters are usually calculated for a small set of zooming factors and considered fixed within those zooming intervals.

4.1.2.2 Extrinsic Parameters

The extrinsic parameters of a camera define the location and orientation of the camera reference frame with respect to a known, world reference, frame. A typical choice for describing the transformation between camera and world frame (see Figure 4.4) is to use a 3D translation vector, $\vec{T}$, describing the relative positions of the origins of the two reference frames, and, an orthogonal 3x3 rotation matrix, R, that brings the corresponding axes of the two frames onto each other.

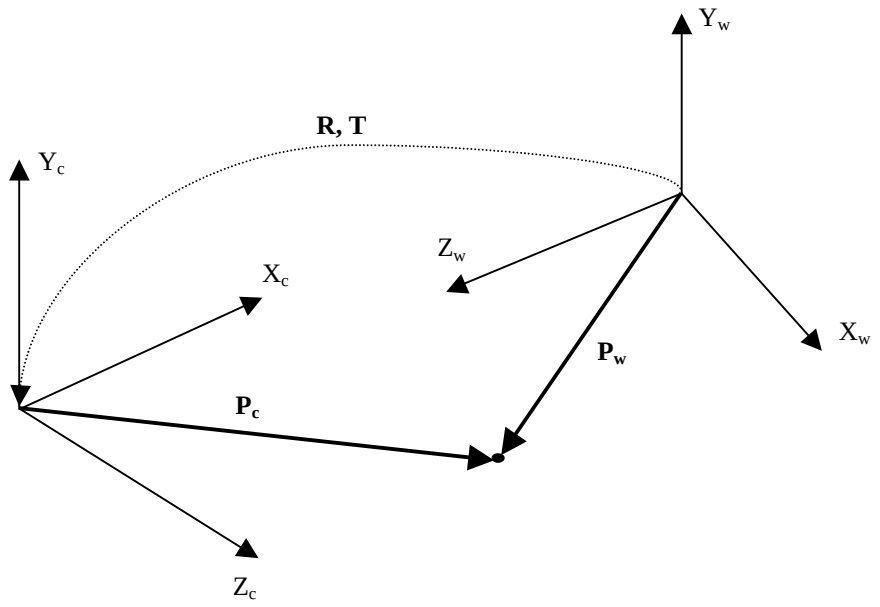


Figure 4.4. Rigid-body transformation.

The transformation between Camera and World Coordinate Systems

The relation between the coordinates of a point *P* in world and camera frame, P_w and P_c, respectively, expressed in homogenous coordinates is:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad \textbf{(4.4)}$$

The 4x4 matrix *[R T]* defines the location and orientation of the camera reference frame with respect to a known world reference frame.

4.1.3 Camera Projection Matrix

The relations linking directly the 2D pixel coordinates of an image point with the 3D world coordinates of the corresponding point, *without explicit reference to the unknown camera reference frame* as in equation (4.2), can now be obtained. The imaging process produced by a projective camera can be interpreted as a sequence of three projective transformations, depicted in Figure 4.5:

1. *Rigid Body Transformation*, as defined by equation (4.4), maps from the world 3D Euclidean coordinate system to the camera 3D coordinate system. The matrix *[R T]* does, in fact, represent the extrinsic parameters of the camera:

$$P_c = \begin{bmatrix} R & T \end{bmatrix} \cdot P_w \quad \textbf{(4.5)}$$

2. *Perspective Projection Transformation*, given by equation (4.2), corresponds to a mapping between the camera 3D Euclidean coordinate system and the image 2D Euclidean coordinate system:

$$p = M_p \cdot P_c \quad \textbf{(4.6)}$$

3. *CCD imaging Transformation*, illustrated by equation (4.3), maps from the image's 2D Euclidean coordinate system to the 2D affine coordinate system of the CCD:

$$w = M_c \cdot p \quad \textbf{(4.7)}$$

Combining equations (4.2) and (4.3), a 3x4 matrix *K* can be obtained which depends only on the intrinsic parameters of the camera:

$$K = M_c \cdot M_p = \begin{bmatrix} k_u f & k_\alpha f & u_0 & 0 \\ 0 & k_v f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \textbf{(4.8)}$$

The overall mapping corresponding to the imaging process is therefore given by the matrix *M* where:

$$M = K \cdot \begin{bmatrix} R & T \end{bmatrix} \quad \textbf{(4.9)}$$

The matrix *M* is called the *projective camera matrix*, and the matrix *K* corresponds to the *matrix of intrinsic parameters*. The matrix *[R T]* is called the *extrinsic or external parameters* matrix. If the matrix *K* is known, the camera is *calibrated*. In-depth studies of

the camera calibration process can be found in [Tsai87], [Faugeras93] or [Zhang00]. TRIP assumes that the intrinsic parameters of a camera supplying frames to a TRIP parser are known or, alternatively, can be easily found. This assumption is valid as several publicly available software packages for camera calibration exist, *e.g.* an implementation of Tsai's method requiring a grid with two orthogonal planes containing square patterns (http://www.cs.cmu.edu/~rgw/TsaiCode.html) or the more convenient software from Zhang (http://research.microsoft.com/~zhang/Calib/), only requiring a planar pattern with black squares over a white background.

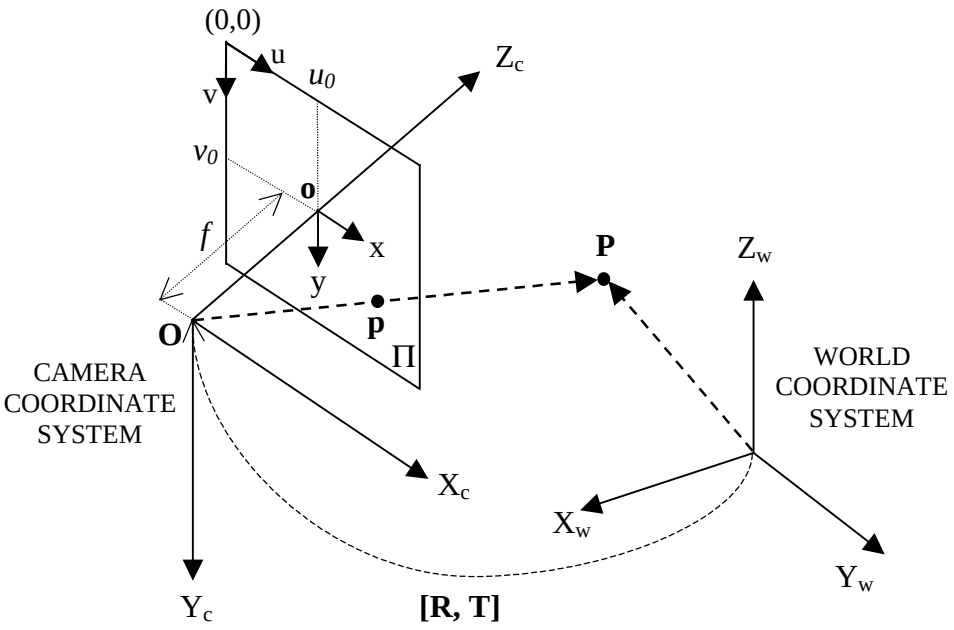


Figure 4.5. The geometry of a pinhole camera.

The Linear Matrix Equation of Perspective Projections

Given a world point $P_w=[X_w, Y_w, Z_w]^T$ and an image point in pixel coordinates $w=[u,v]^T$, the perspective projection, M, of P_w on w is given by:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = M \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad \textbf{(4.10)}$$

4.2 Model-Based Location

Model-based object location is the task of finding the pose in space of previously identified objects (of known in geometry) in an image. The term *pose* means the position and orientation of an object with regards to a viewing camera, *i.e.* the translation and rotation that bring the object centred coordinate system into the sensor centred coordinate system.

Model-based Object Location

Given an image, a sensor model and the geometric model of the object imaged, find the 3D position and orientation of the model that generated the image.

Model-based object location usually assumes the following:

1. Location is extracted from a single intensity image
2. A model of the geometric image formation (sensor model) is completely known, *i.e.* the camera intrinsic parameters are known.
3. Object models are object-centred, and based on geometric features. All model points and vectors are expressed in the *model reference frame* (or *world coordinate system).*
4. The object identification problem has already been solved, which means that all significant data points and vectors are expressed in the *image reference frame*.

4.3 Pose Extraction using TRIPtags

The TRIP sensor system applies a model-based object location technique to a single intensity image in order to extract the pose of TRIP-tagged objects with respect to a viewing camera. The 'TRIP object model' (see Figure 4.6) is composed of: (1) the three concentric circular borders, of known radius, of the central bull's-eye of a TRIPtag (solid circular borders in the figure) and (2) a point outside the bull's-eye (denoted by P in Figure 4.6), lying on the X-axis of the target centred coordinate system. This point corresponds to the bottom outermost corner of the synchronisation sector of a TRIPtag. In fact, only the point P and the outermost circular border of the central bull's-eye are used,

since the outer border can be determined most accurately from the image, *i.e.* more pixels are available to fit an ellipse.

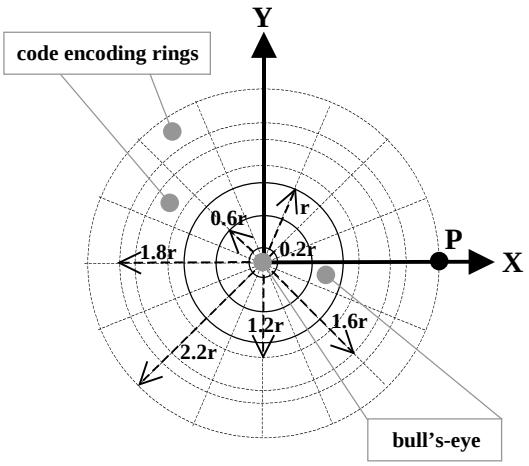


Figure 4.6. TRIPtag geometric model.

POSE_FROM_TRIPTAG

Given a *pre-calibrated* camera's single view of a TRIPtag of known size, the solution sought is the translation vector $\vec{T} = (T_x, T_y, T_z)$ and rotation angles (α, β, γ) that define the rigid body transformation between a camera coordinate system and a target centred coordinate system.

The transformation sought is called *rigid body transformation* because it preserves the size of the object. The problem is equivalent to determining the camera's *extrinsic parameters* with respect to the target centred reference frame. The proposed method, named `POSE_FROM_TRIPTAG`, is explained in the following sections.

4.3.1 POSE_FROM_TRIPTAG overview

The `POSE_FROM_TRIPTAG` method is applied to the outcome of the target recognition process described in chapter 3. The pose extraction method takes as input the following parameters: (1) the parameters of the conic equation representing the outermost elliptical border image of an identified target's bull's-eye, (2) the pixel coordinates of the image of the bottom outermost corner of a TRIPcode's synchronisation sector (marked by a small

triangle at 3 o'clock in Figure 4.7), (3) the radius of the bull's-eye of the sighted target. This radius is encoded in the four sectors reserved for this purpose in a TRIPtag. Based on that data, the method establishes a *homography*, *i.e.* a projective transformation from one plane to another, that back-projects the elliptical outermost border of a target image into its actual circular form in the target reference plane. As a result of this, the translation vector $\vec{T} = [T_x, T_y, T_z]$ and rotation angles (α, β, γ) that define the rigid body transformation between a camera coordinate system and a target centred coordinate system are returned. An important assumption of this procedure is that the camera intrinsic parameters must be known, *i.e.* the camera used must have been calibrated. The `POSE_FROM_TRIPTAG` method was inspired by and extends the `POSE_FROM_CIRCLE` method described in [Forsyth+91].

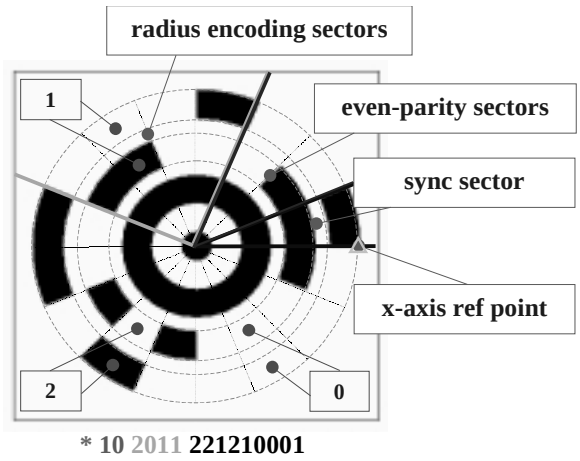


Figure 4.7. TRIPcode of radius 58mm and ID 18,795.

Figure 4.8 and Figure 4.9 show the geometry of the `POSE_FROM_TRIPTAG` method. The ellipse corresponding to the image of the outermost circular border of the bull's-eye of a target defines a cone with its vertex at the centre of projection of the pinhole camera (O). The orientation of the target's plane, Π_t, is found by rotating the camera so that the intersection of the cone with the image plane becomes a circle, which happens when the image plane, Π_i, is parallel to the target plane, Π_t. This rotation can be estimated as the composition of two successive rotations. The first, see Figure 4.8, is a 3D rotation that puts the Z axis through the centre of the target, and aligns the X and Y axes with the axes

of the image ellipse; the second, see Figure 4.9, rotates the newly found coordinate system around *Y'* until the image plane becomes parallel to Π_t.

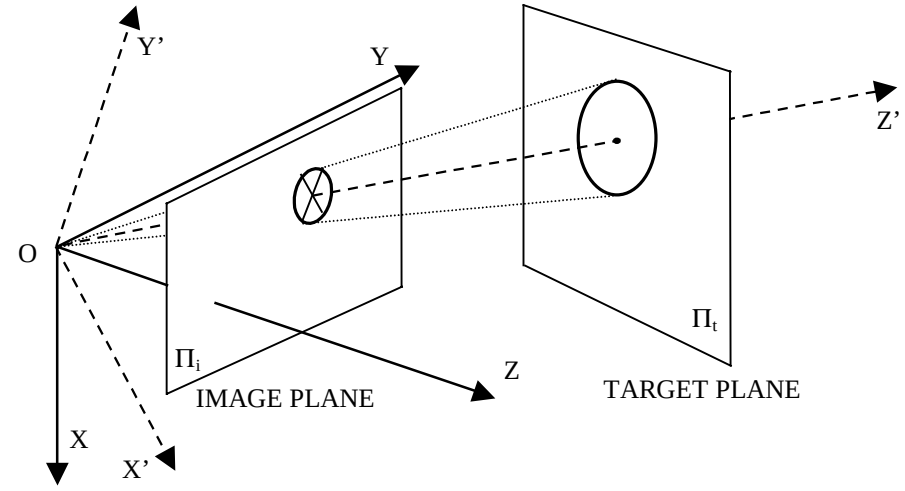


Figure 4.8. 3D rotation of *(X,Y,Z)* to the eigenvector frame *(X', Y', Z')*.

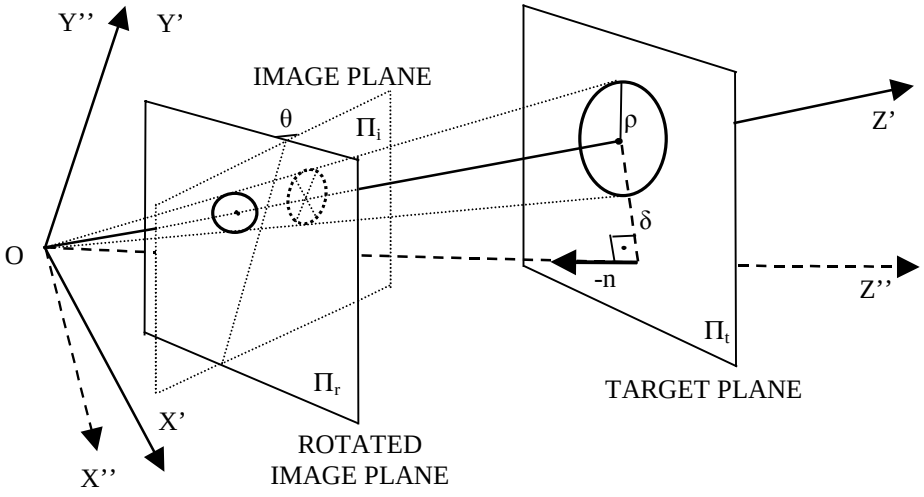


Figure 4.9. Rotation of plane Π_i about the *Y'* axis by an angle θ.

The `POSE_FROM_TRIPTAG` method exploits the property, unique to a conic (see equation (3.6)) that is a circle, that the back-projected curve must have equal coefficients x^2 and y^2 and no term in *xy*. Four stages are required for the determination of the six degrees of freedom (three components of the translation vector plus the three angles defining the 3D

rotation around the three axes of the target centred coordinate system) that define the rigid body transformation aligning the camera and target model reference frames:

1. Target plane orientation estimation.
2. Target 3D location estimation.
3. Breaking the ambiguity from the two-fold solution provided by the method.
4. Correcting rotation angle miscalculations due to circle symmetry.

These stages are discussed in the following four sections. Appendix A summarises the entire set of operations necessary to extract the pose from a single view of a TRIPtag. Figure 3.14 shows the outcome of applying the `POSE_FROM_TRIPTAG` method to an image depicting a user wearing a TRIP marker.

4.3.2 Target Plane Orientation

Let ellipse $ax^2+bxy+cy^2+dx+ey+f = 0$ be the image of the outermost border of a target's bull's-eye taken as reference. Then the image of the curve defines a cone in 3D, given by:

$$ax^2+bxy+cy^2+dxz+eyz+fz^2 = P^TCP = 0 \quad \textbf{(4.11)}$$

where $P=[x, y, z]^T$ is a point in the cone and C is the real symmetric matrix of a cone:

$$C = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \quad \textbf{(4.12)}$$

This method assumes that the reference ellipse is expressed in the camera coordinate system rather than in the image plane pixel coordinates as returned by the target recognition process. Thus, the image origin must be at the principal point (the intersection between the optical axis and the image plane), and the distances are measured in units of focal length, which is set to 1. To achieve this, the real symmetric matrix C (expressed in CCD array pixel coordinates) is normalised by using the matrix K (see equation (4.8)) of intrinsic camera parameters:

$$C_n = K^T \cdot C \cdot K \quad \textbf{(4.13)}$$

The orientation of the circle's plane, Π_t, is found by rotating the camera so that the intersection of the cone with the image plane becomes a circle, which happens when the image plane, Π_i, is parallel to the target plane. This rotation is estimated as the composition of two successive rotations, namely R_1 and R_2:

$$R_C = R_1 \cdot R_2 \quad \textbf{(4.14)}$$

The first rotation, R_1, is determined by diagonalising C_n, *i.e.* removing the coefficients with terms in *xy*, *xz* and *yz*. This 3D rotation (see Figure 4.8) transforms the camera coordinate system, *OXYZ*, to the eigenvector frame, $OX'Y'Z'$, and the ellipse matrix C_n into C'. If λ_1, λ_2, λ_3 are the eigenvalues of C_n, with $\lambda_1 < \lambda_2 < \lambda_3$ and $\vec{e_1}$, $\vec{e_2}$, $\vec{e_3}$, the corresponding eigenvectors, then:

$$\overrightarrow{P'} = R_1^T \cdot \vec{P}$$

$$R_1 = \left[\vec{e_1} \quad \vec{e_2} \quad \vec{e_3}\right]$$

$$C' = R_1^T \cdot C_n \cdot R_1 = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

(4.15)

The second rotation, R_2, is obtained by imposing the equality of the coefficients of x^2 and y^2 in C'. R_2 represents a rotation around the Y' axis by an angle θ, as shown in equation (4.16). This rotation, R_2, sends a point P' to P'' and transforms C' into C'' (see Figure 4.9):

$$R_2 = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad \textbf{(4.16)}$$

$$C'' = R_2^T \cdot C' \cdot R_2 = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} =$$

$$\begin{bmatrix} \sin^2\theta \cdot \lambda_3 + \cos^2\theta \cdot \lambda_1 & 0 & \sin\theta \cdot \cos\theta(\lambda_1 - \lambda_3) \\ 0 & \lambda_2 & 0 \\ \sin\theta \cdot \cos\theta(\lambda_1 - \lambda_3) & 0 & \sin^2\theta \cdot \lambda_1 + \cos^2\theta \cdot \lambda_3 \end{bmatrix} \quad \textbf{(4.17)}$$

$$\overrightarrow{P''} = R_2^{\ T} \cdot \overrightarrow{P'} = R_2^T \cdot R_1^T \cdot \vec{P} \quad \textbf{(4.18)}$$

Given that the coefficients of x^2 and y^2 must be equal for C'' to be a circle and considering equation (4.17), an expression for the angle θ can be inferred:

$$\sin^2\theta \cdot \lambda_3 + \cos^2\theta \cdot \lambda_1 = \lambda_2$$

$$\sin^2\theta \cdot \lambda_3 + (1 - \sin^2\theta) \cdot \lambda_1 = \lambda_2$$

$$\sin^2\theta = \frac{\lambda_2 - \lambda_1}{\lambda_3 - \lambda_1} \quad \textbf{(4.19a)}$$

$$\cos^2\theta \cdot \lambda_1 + (1 - \cos^2\theta) \cdot \lambda_3 = \lambda_2$$

$$\cos^2\theta = \frac{\lambda_3 - \lambda_2}{\lambda_3 - \lambda_1} \quad \textbf{(4.19b)}$$

$$\theta = \pm \arctan \sqrt{\frac{\lambda_2 - \lambda_1}{\lambda_3 - \lambda_2}} \quad \textbf{(4.20)}$$

The composite rotation R_C, which is the result of multiplying R_1 and R_2, transforms the image plane, Π_i, so that it becomes parallel to the target plane, Π_t (see Figure 4.9). Consequently, a vector normal to the target plane can be obtained by applying equation (4.21). Vector $\vec{n}$ represents the orientation of the plane Π_t expressed in the camera coordinate system.

Target Orientation

$$\vec{n} = R_C \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T = \begin{bmatrix} R_{13} & R_{23} & R_{33} \end{bmatrix}^T \quad \textbf{(4.21)}$$

As result of equation (4.20), there is a two-fold ambiguity in the recovered orientation depending on which sign of θ is chosen, *i.e.* $\vec{n}_1$ from R_{C1} and $\vec{n}_2$ from R_{C2}. Section 4.3.4 explains the geometric implications of this result and a way to break this ambiguity.

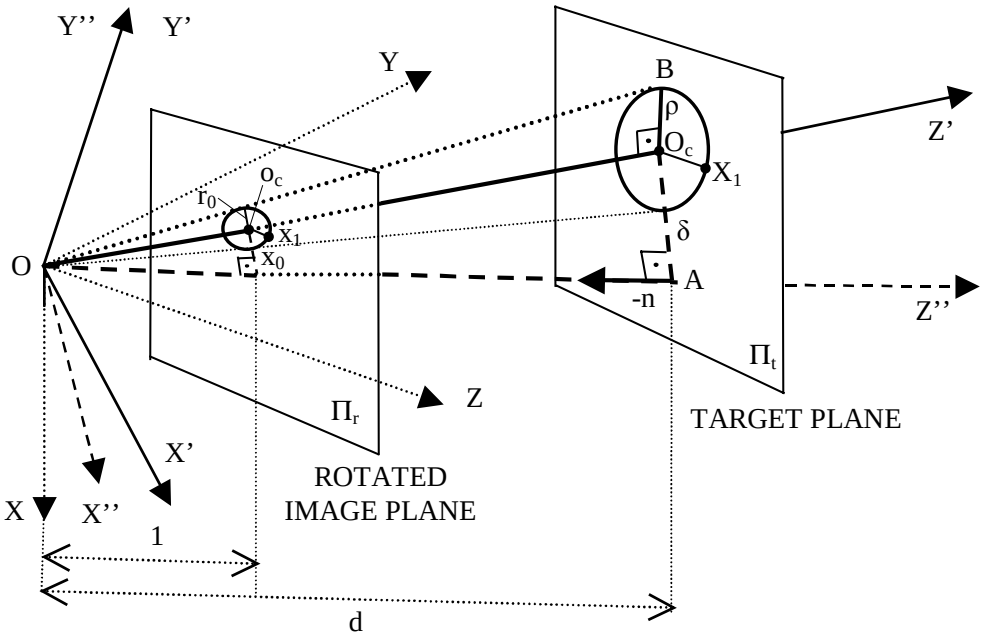


Figure 4.10. Geometric relations between planes Π_r and Π_t.

4.3.3 Target 3D Location

C'' represents the real symmetric matrix of a circle in the image plane of radius r_0 and centre (x_0, *0, 1*), in terms of the coordinate system $OX''Y''Z''$, as illustrated in Figure 4.10. Equation (4.17) can, therefore, be simplified making the terms x^2 and y^2 equal to λ_2, and then dividing by λ_2, as depicted in equation (4.22).

$$C''=\begin{bmatrix}\lambda_2 & 0 & \Delta\\ 0 & \lambda_2 & 0\\ \Delta & 0 & \Phi\end{bmatrix}=\begin{bmatrix}1 & 0 & \Delta/\lambda_2\\ 0 & 1 & 0\\ \Delta/\lambda_2 & 0 & \Phi/\lambda_2\end{bmatrix}=\begin{bmatrix}1 & 0 & -x_0\\ 0 & 1 & 0\\ -x_0 & 0 & x_0^{\,2}-r_0^{\,2}\end{bmatrix} \quad \textbf{(4.22)}$$

Equating correspondences between equation (4.17), equations (4.19a) and (4.19b), and the first equality in equation (4.22), it can be derived that:

$$\Phi = \sin^2\theta \cdot \lambda_1 + \cos^2\theta \cdot \lambda_3 = \lambda_1 - \lambda_2 + \lambda_3 \quad \textbf{(4.23)}$$

$$\Delta^2 = (\sin\theta \cdot \cos\theta(\lambda_1 - \lambda_3))^2 = (\lambda_2 - \lambda_1)(\lambda_3 - \lambda_2) \quad \textbf{(4.24)}$$

If the same process to correspondences in the last two equalities of equation (4.22) is now applied, the values for the coordinate X'', x_0, and the radius of the centre of the imaged circle C'', r_0, are obtained.

$$x_0{}^2 = \frac{\Delta^2}{\lambda_2^2} = \frac{(\lambda_2 - \lambda_1)(\lambda_3 - \lambda_2)}{\lambda_2^2} \quad \textbf{(4.25)}$$

$$r_0{}^2 = x_0{}^2 - \Phi / \lambda_2 = \frac{-\lambda_3 \cdot \lambda_1}{\lambda_2^2} \quad \textbf{(4.26)}$$

The distance between the camera and the target plane, denoted by d in Figure 4.10, can be obtained by applying triangle similarity principles to the triangle OO_cA-Oo_cx_o:

$$\frac{d}{1} = \frac{\delta}{x_0} \quad \textbf{(4.27)}$$

From triangles OO_cB and Oo_cr_0, and applying equations (4.25), (4.26) and (4.27), an expression for δ in terms of the eigenvalues of C_n and the known radius of the outermost border of a target's bull's-eye, ρ, can be derived (see equation (4.28)). Similarly, substituting equation (4.28) in (4.27), an expression for the distance of the target plane to the camera origin, d, is obtained (see equation (4.29):

$$\frac{\rho}{r_0} = \frac{\delta}{x_0} \Rightarrow \delta = \sqrt{\frac{-(\lambda_2 - \lambda_1) \cdot (\lambda_3 - \lambda_2)}{\lambda_1 \cdot \lambda_3}} \cdot \rho \quad \textbf{(4.28)}$$

$$d = \frac{\rho}{r_0} = \sqrt{\frac{-\lambda_2^2}{\lambda_1 \cdot \lambda_3}} \cdot \rho \quad \textbf{(4.29)}$$

The 3D coordinates of the centre of the target (see Figure 4.10), expressed in the $OX''Y''Z''$ frame, correspond to the translation vector, $\vec{T}$, which can be calculated in terms of the original coordinate system *OXYZ* as:

Target 3D Location

$$\vec{T} = R_C \cdot [\delta \quad 0 \quad d]^T \quad \textbf{(4.30)}$$

Again, as a consequence of equation (4.20), there are two possible solutions for the translation vector, *i.e.* $\vec{T_1}$ .and $\vec{T_2}$. Appendix B proves that, in fact, solutions $\vec{T_1}$ .and $\vec{T_2}$ are almost identical and could, therefore, be used interchangeably.

4.3.4 Breaking the ambiguity

As noted by the previous two sections, the `POSE_FROM_TRIPTAG` method returns two feasible pose solutions. This result can be explained geometrically by observing Figure 4.11. Circle 1 and circle 2, lying on planes Π_1 and Π_2, respectively, and with the same radius *r* are projected to the same ellipse in the image plane. Consequently, it is necessary to use additional information, apart from the ellipse of reference, in order to be able of deciding which of the two solutions is the real one. For TRIP, this additional information is provided by the known projection of the bottom outermost corner of the synchronisation sector of a TRIPtag (see Figure 4.7). The projection of this point is accurately identified by the code deciphering stage of the target recognition process, described in section 3.4.7. Likewise, the location of this point in the target coordinate system is also known (see Figure 4.6). If the projections of this point, obtained by means of the two rigid body transformations returned by the `POSE_FROM_TRIPTAG` method, are compared with the known projection of this point in the image, the right solution can be determined. The calculated projection lying closest to the known projection in the image will correspond to the right solution. Figure 4.12 shows how the projection of a point at a distance $r+d$ from the centre of a circle in plane Π_1 will differ significantly from the projection of the same point lying on plane Π_2.

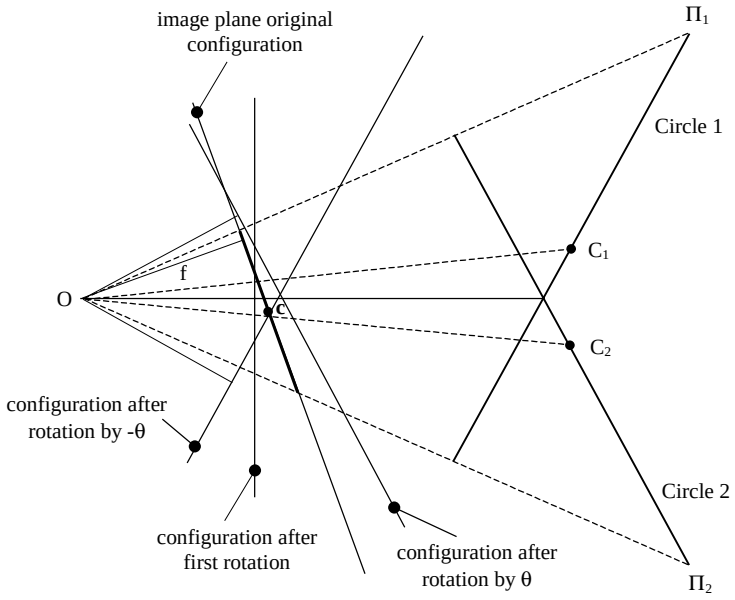


Figure 4.11. Circle projection ambiguity.

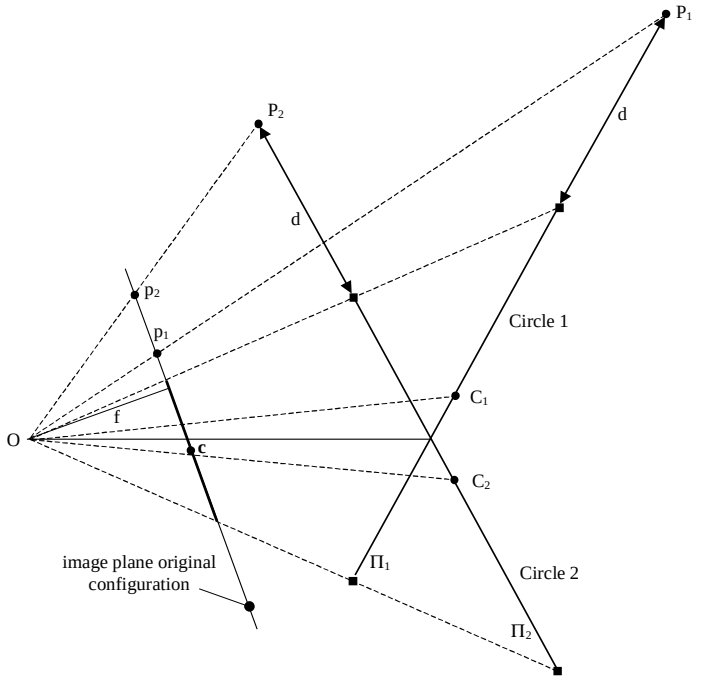


Figure 4.12. Projections of points P_1 and P_2 at $r+d$ distance from C_1 and C_2.

The projection of point $P=[2.2 \cdot r,\ 0,\ 0,\ 1]^T$ in the TRIPtag plane into point $p=[su,\ sv,\ s]^T$ in the image plane can be obtained by applying the *planar projective transformation* [Faugeras93]:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K \cdot \begin{bmatrix} R_{C_{11}} & R_{C_{12}} & T_x \\ R_{C_{21}} & R_{C_{22}} & T_y \\ R_{C_{31}} & R_{C_{32}} & T_z \end{bmatrix} \cdot \begin{bmatrix} 2.2 \cdot r \\ 0 \\ 1 \end{bmatrix} \quad \textbf{(4.31)}$$

The planar projective transformation exploits the fact that every point lying on the target plane has a value of *Z=0*. Equation (4.31) can be applied to the two feasible results obtained, *i.e.* (R_{C1} and $\vec{T}_1$) and (R_{C2} and $\vec{T}_2$), so as to calculate the projections p_1 and p_2. In order to measure the distances between p_1 and p (the known projection of the point in the image), *and* p_2 and p, the homogeneous coordinates of these points are transformed into Cartesian coordinates, as explained in Figure 4.2. The right solution should project P_i into a point lying very close to p:

Choosing the Right Solution

$$\vec{T}, n = \begin{cases} \vec{T}_1, n_1 & if \quad \|p_1 - p\| < \|p_2 - p\| \\ \vec{T}_2, n_2 & if \quad \|p_1 - p\| \geq \|p_2 - p\| \end{cases} \quad \textbf{(4.32)}$$

4.3.5 Correcting the Rotation Angles

The use of a circle and a point outside the circle as a reference has provided a closed form solution to the determination of the 3D pose of a TRIPtag. Unfortunately, due to the circle symmetry, the view of a planar circle does not permit the determination of rotations around the *Z*-axis of the target coordinate system, orthogonal to the target's plane. No matter how much a circle is rotated around the *Z*-axis, its projected image looks the same. This means that from the rotation matrix R_C, only the angles (α, β) around the axes *X* and *Y* can be obtained. Therefore, it is necessary to calculate a new rotation matrix, named R_C' from which the angle γ around the *Z*-axis can also be recovered.

The vector columns of the R_C' matrix are, in fact, the three unit vectors defining the TRIP target-centred coordinate system. From these vectors, the one corresponding to the target's *Z*-axis is already known and given by $\vec{n}$. In order to calculate the other two column vectors, namely $\vec{r}_x$ and $\vec{r}_y$, is necessary to use two correspondences between

points expressed in the target's frame and their projections in the image plane. The first point correspondence is given by the known centre of the target, $\vec{T}$, and its projection in the image. The second correspondence used is the back-projection of the bottom outermost corner of the synchronisation sector projection, denoted by a small triangle in Figure 4.7 and named x_1 in Figure 4.10, into X_1. This point can be uniquely identified in every projection of a TRIPtag. The following is applied to calculate this correspondence.

Given a vector $\vec{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$, any other vector with the same direction and origin is given in homogenous coordinates by equation (4.33), where s indicates a free scale factor applied to the vector's modulus:

$$\overrightarrow{P(s)} = \begin{bmatrix} \vec{p} \\ s \end{bmatrix} \quad \textbf{(4.33)}$$

Therefore, the correspondence between the 3D point X_1, and its projection in the image x_1 is given by:

$$\overrightarrow{X_1} = \begin{bmatrix} \overrightarrow{Ox_1} \\ s_c \end{bmatrix} = \begin{bmatrix} x_{1x} \\ x_{1y} \\ x_{1z} \equiv 1 \\ s_c \end{bmatrix} \quad \textbf{(4.34)}$$

The scale factor s_c can be determined by considering that the point X_1 belongs to the target plane Π_t, *i.e.*:

$$\overrightarrow{P(s)}^T \cdot \Pi_t = 0 \quad \textbf{(4.35)}$$

where Π_t is given by the *point normal plane* equation [Farin+97]. This equation states that given a point p and a normalised vector $\vec{n}$, *i.e.* $\|n\| = 1$, bound to p, a plane is defined by the locus of all points x that satisfy the equation:

$$\vec{n}\cdot(\vec{x}-\vec{p})=0 \Rightarrow n_1x+n_2y+n_3z-(n_1p_1+n_2p_2+n_3p_3)=0 \Rightarrow$$
$$Ax+By+Cz+D=0 \quad \textbf{(4.36)}$$

where $A = n_1$, $B = n_2$, $C = n_3$ and $D = -(n_1p_1+n_2p_2+n_3p_3)$. It can be proved that $|D|$ reflects the distance of the plane to the coordinate origin. Therefore, based on equation (4.36), the equation of plane Π_t can be represented in homogenous form as:

$$\Pi_t=\begin{bmatrix}x & y & z & 1\end{bmatrix}\cdot\begin{bmatrix}\vec{n}\\ -d\end{bmatrix}=0 \quad \textbf{(4.37)}$$

where $\vec{n}$ is the previously calculated orientation of the target plane with regard to the camera and d represents the calculated distance from the camera centre of projection to the plane.

Thus, substituting equation (4.37) into (4.35), and using the known coordinates of the bottom outermost corner of the synchronisation sector of the TRIPtag (x_{1x}, x_{1y}), the value of s_c is:

$$s_c=\frac{\overrightarrow{o_c^T}\cdot\vec{n}}{d} \quad \textbf{(4.38)}$$

Finally, the Cartesian coordinates of O_c, expressed in the original coordinate system, *OXYZ*, are calculated by combining equations (4.38) and (4.34):

$$\overrightarrow{X_1}\equiv\begin{bmatrix}x_{1x}/s_c\\ x_{1y}/s_c\\ (x_{1z}\equiv 1)/s_c\end{bmatrix} \quad \textbf{(4.39)}$$

The first column of the rotation matrix $\vec{r}_x$ can hence be calculated as the unitary vector joining $\vec{T}$ and $\overrightarrow{X_1}$ in Figure 4.10, *i.e.*:

$$\vec{r}_x = \frac{(X_1 - O_c)}{\|X_1 - O_c\|} \quad \textbf{(4.40)}$$

The second column of the rotation matrix, $\vec{r}_y$, can be estimated by considering that the columns of a rotation matrix, or the unitary vectors defining a coordinate system, are orthogonal. Therefore, a third orthogonal vector can be obtained by taking the cross product of the other two:

$$\vec{r}_y = \vec{n} \times \vec{r}_x \quad \textbf{(4.41)}$$

Thus, the rotation matrix defining the transformation between OXYZ and OX″Y″Z″ is:

$$R_C' = \left[\vec{r}_x \quad \vec{r}_y \quad \vec{n}\right] \quad \textbf{(4.42)}$$

The matrix R_C' is a 3D rotation resulting from the composition of three consecutive rotations, namely R_x, R_y, R_z, around the coordinate axes, X, Y, Z, by angles α, β and γ respectively. The form of R_C' is given by (4.43):

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \quad R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \quad R_z = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_x \cdot R_y \cdot R_z =$$

$$\begin{bmatrix} \cos\beta\cdot\cos\gamma & -\cos\beta\cdot\sin\gamma & \sin\beta \\ \sin\alpha\cdot\sin\beta\cdot\cos\gamma + \cos\alpha\cdot\sin\gamma & -\sin\alpha\cdot\sin\beta\cdot\sin\gamma + \cos\alpha\cdot\cos\gamma & -\sin\alpha\cdot\cos\beta \\ -\cos\alpha\cdot\sin\beta\cdot\cos\gamma + \sin\alpha\cdot\sin\gamma & \cos\alpha\cdot\sin\beta\cdot\sin\gamma + \sin\alpha\cdot\cos\gamma & \cos\alpha\cdot\cos\beta \end{bmatrix}$$

(4.43)

Given (4.42) and (4.43), the angles α, β and γ can be obtained. The angle β, to be visible by the camera, must be between $[-\pi/2, \pi/2]$ and can be calculated directly from $R_{C'13}$ as:

Rotation Around Y-axis of Target

$$\beta = \arcsin(R_C{}'_{13}) \quad \textbf{(4.44)}$$

The angle α can also be directly obtained from the third column (*i.e.* vector $\vec{n}$) of R_C'. Again, to be visible this angle is constrained to values in the range [-π/2, π/2]:

Rotation Around X-axis of Target

$$\alpha = \arcsin\left(\frac{-R_C{}'_{23}}{\cos\beta}\right) \quad \textbf{(4.45)}$$

Finally the angle γ is calculated using $R_C'_{12}$ and $R_C'_{11}$:

Rotation Around Z-axis of Target

$$\gamma_1 = \arcsin(-R_C{}'_{12} / \cos\beta)$$
$$\cos\gamma_2 = R_C{}'_{11} / \cos\beta \quad \textbf{(4.46)}$$

If both $cos\gamma_1$ and $cos\gamma_2$ have the same sign then γ is set to γ_1, otherwise γ is set to $\pi-\gamma_2$.

The angles α, β and γ determine the orientation of a TRIP target with respect to the viewing camera.

4.4 TRIP Sensor Adaptive Operation

The main limitation of vision-based sensors is their high processing demands. TRIP, despite being optimised to require the least possible processing load, is not an exception and still presents relatively high processing demands. Nevertheless, a TRIP sensor will often be analysing frames where no TRIPtags are encountered, or alternatively, frames where the approximate location of a TRIPtag in the image could be inferred based on its location in previous frames. Therefore, it is convenient to provide the TRIP sensor with some intelligence both to distinguish the worthiness of analysing an image, and if the analysis proceeds, to determine whether full-image parsing should take place, or if this process should just be applied to a smaller sub-window in the image. The aim is to give TRIP maximum responsiveness (real-time) but still guarantee the minimum consumption of processing load. For this reason, TRIP is provided with an *adaptive* behaviour during

its operation. The sensor transitions through three different operating modes, named *default-mode*, *saving-mode* and *real-time-mode*, depending on the characteristics of the scene viewed. Figure 4.13 shows a state diagram with the different operation modes of the TRIP sensor and the conditions upon which the sensor switches from one to another.

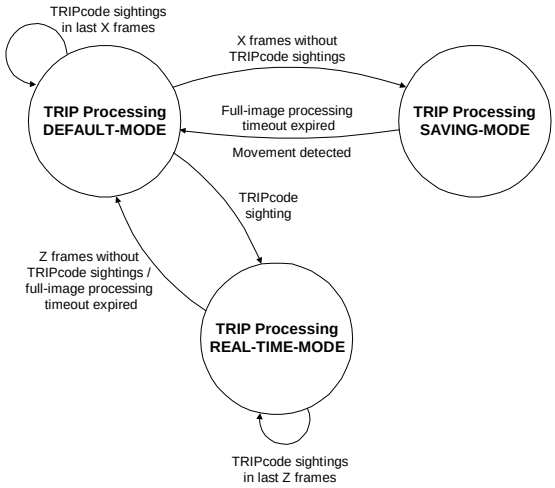


Figure 4.13. TRIP sensor's operating modes diagram

The TRIP default-mode of operation is activated either on sensor start-up or whenever a timeout set to guarantee full-image analysis expires. The timeout, specified as a parameter in the system bootstrap, limits the time the sensor spends in either the saving or real-time modes, where full-image analysis does not take place. This is necessary to ensure a good responsiveness of the system to scene changes.

The TRIP saving-mode is triggered by the default-mode whenever there are no TRIPcode sightings in a given number of frames specified at the system start-up. This mode undertakes a *Triggering Analysis* by which TRIP decides whether it should return to the default mode (if significant changes in the scene have occurred) or otherwise stop consuming processing resources, *i.e.* sleep. Basically, an efficient movement detection mechanism is implemented. Each time slot the triggering module analyses a low-resolution version of a newly captured image (*i.e.* one in every 5 pixels in both horizontal and vertical directions) and compares each pixel with a previously calculated *Running Video Average* (RVA) frame of the background. An RVA [Stafford-Fraser96] is a method used to construct an evolving background frame, insensitive to small variations in

illumination or the slight flicker of electric light, using the average pixel values of the N preceding images. The triggering stage determines the percentage P of pixels that have changed by more than a certain threshold. A high percentage means that movement has been detected and the parsing control is then passed to the default-mode. Otherwise the TRIP sensor is set to sleep for a short period and then the same triggering analysis process is repeated. When the full image processing timeout expires, control is passed back to the default mode.

The TRIP real-time-mode of operation is triggered by the default-mode each time a TRIPcode is spotted in an image. This mode exploits the spatial locality of TRIPtag images in successive video frames. Once a target has been identified within an image, its outermost elliptical border is tracked in subsequent frames. The ellipse tracking method undertakes edge detection along the normal to the tangents of the previous ellipse at various points, in order to identify the new location of points belonging to the new ellipse. Then, the method reapplies the ellipse-fitting algorithm with the newly found ellipse points. Finally, the `POSE_FROM_TRIPTAG` method is applied to the newly obtained ellipse parameters. In order to guarantee the rapid identification of newly appearing TRIPtags, when the full image processing timeout expires, control is passed back to the default mode. Furthermore, if after Z frames no TRIP sightings are found then control is also returned to the default-mode.

4.5 TRIP Sensor Accuracy and Performance

This section illustrates the accuracy and performance measurements corresponding to a C++ implementation of the video frame parsing process of the TRIP system.

4.5.1 POSE_FROM_TRIPTAG Accuracy

In order to estimate the 3D location accuracy provided by the `POSE_FROM_TRIPTAG` method, a comparison was performed between the results returned by TRIP and the ones returned by a third party calibration software, implementing Tsai's method [Tsai87]. As shown in Figure 4.14, the centre of a TRIPtag (with a size of 66x66 mm^2) was attached to one of the corners of a set of black squares on a white background constituting a 3D

calibration pattern. Based on the high number of corner correspondences (184)[15] that can be established between the calibration grid of known geometry and its projection in an image, the camera calibration software recovers the extrinsic parameters of the viewing camera with very high accuracy[16]. The pose calculated by this method was compared with the results returned by the `POSE_FROM_TRIPTAG` method.

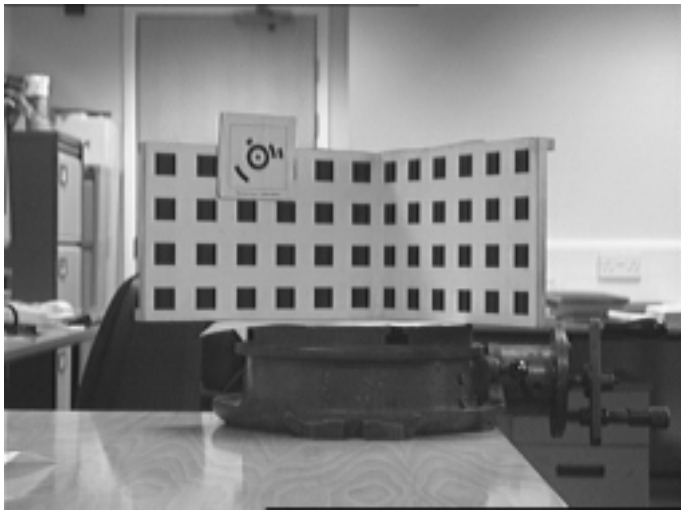


Figure 4.14. Picture of a camera calibration grid with TRIPtag.

The calibration grid with the TRIPtag attached at a known square corner location (see Figure 4.14) was located at distance intervals of 10 cm ranging between 120 cm and 190 cm from the viewing camera. Likewise, at each fixed distance, *e.g.* 130 cm, incremental slants by 10 degrees in the range 20 to 60 degrees, between the normal to the TRIPtag plane and the camera's *Z*-axis direction were considered. The results obtained when analysing 768x576 pixel images of the grid at the different distances and slants from the viewing camera are shown in Figure 4.15 and Figure 4.16.

[15] 192 – 8 occluded by the TRIPtag.

[16] Some practical measurements with a tape and a protractor were undertaken in order to verify the high accuracy with which the Tsai's method implementation returns the position of tagged objects.

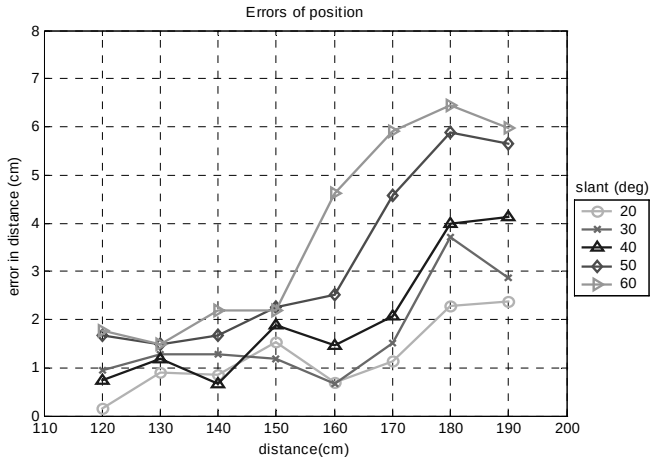


Figure 4.15. Errors of position.

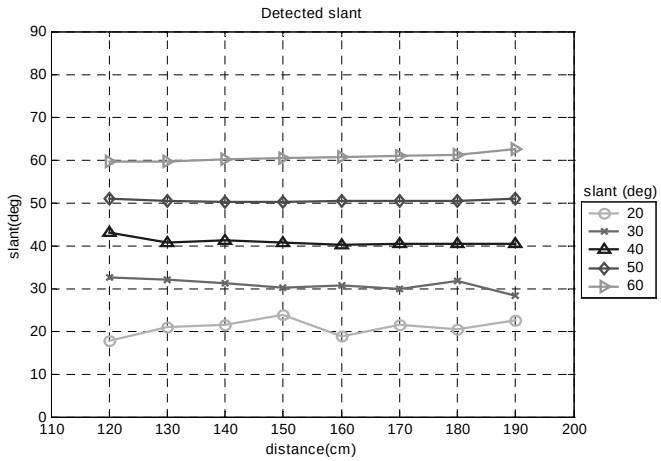


Figure 4.16. Detected slant.

Figure 4.15 shows the obvious fact that the further the target is from the camera and the more slant that exists between a normal to its plane and the camera's *Z*-coordinate axis, the worse the distance estimation from the camera to the target performed by the pose extraction method becomes. False negatives occur when the slant presented by a TRIPtag is above the 70 degrees threshold, or the target is too far from the camera (3 metres or more for the camera used). The distance is estimated with less than a 3% error for distances below 160 cm, providing the slant between the target plane and the camera *Z*-coordinate axis is below 70 degrees. From the 160 cm distance onwards, the error percentage is higher. This is due to the fact that the pixel areas occupied by the projections of the targets are too small. These experiments demonstrate that whenever a target is placed with a slant below 70 degrees and its projection occupies a pixel area of at

least 1200 pixels (approximately a region of 40x40 pixels, where the bull's-eye's projection occupies approximately 20x20 pixels), the accuracy of the location information provided by TRIP is very good (error < 3%). Figure 4.16 shows that the slant estimation returned by the `POSE_FROM_TRIPTAG` method is rather precise and lies within a 2% range of its real value. This implies that the slant estimation is independent on the severity of the actual target slant.

The accuracy in the position and orientation returned by TRIP may be influenced by the image noise. Noise can be produced by many factors such as light intensity, type of camera and lens, motion, temperature, dust, and others.

Compared with the CyberCode [Rekimoto+00] and ARToolKit [Kato+99] barcode-based position extraction systems, TRIP appears to be more suitable for the recognition and location of markers at a further distance from a camera. Fewer pixels in the images captured seem to be necessary to accurately identify and locate tagged objects in 3D space. Unfortunately, only accuracy results for the ARToolKit system have been encountered with which to compare TRIP's results. The reported ARToolKit's results are for distances to the camera in the range 10-60 cm. In this range, its location and orientation results are very similar in accuracy as the ones reported by TRIP at a much further distance range, *i.e.* 120-190 cm. Hence, TRIP is capable of determining the pose and identifier of targets at greater ranges than seems possible with the ARToolKit. Moreover, the number of identifiers that can be uniquely represented is bigger. In the ARToolKit, the identity encoding patterns are compared by template matching with other patterns given to the system previously. Therefore, these patterns have to differ significantly from each other. In contrast, CyberCode's address space is bigger than TRIP's one. However, CyberCode's complex pattern appears to require a higher image resolution for accurate pose extraction and identification. Objective accuracy comparisons could only be established if results were reported for CyberCode and ARToolKit indicating how many pixels in an image a marker must occupy for accurate recognition and location. Another factor of interest in a comparison study would be the facility with which the markers could be identified in a cluttered environment.

4.5.2 TRIP Sensor Performance

Experiments to evaluate the performance of the TRIP sensor system were also undertaken, using for our tests the 640x480 pixel image shown in Figure 4.17. This image was processed 100 times in a computer with a 1.4 GHz AMD Athlon processor and 512 Mbytes of RAM. The average time to process an image in this machine was 65.0138 milliseconds, which implies that 15.381 (≈ 16) frames per second (fps) can be processed. This 16 fps processing rate, insufficient for undertaking real-time video parsing, is increased to real-time through the adaptive behaviour operation described in section 4.4. In this way, the performance of the TRIP system can be compared with the real-time behaviour provided by the CyberCode and ARToolKit systems.

Ellipse params:
x (495.31), y (427.70) pixel coord
a (11.68), b (10.22) pixel coord
θ (6.54) degrees

Bull's-eye's radius: 0120 (15 mm)
TRIPcode: 002200000 (1,944)

Translation Vector (meters):
T_x=0.3032, T_y=0.1317, T_z=1.9780

Plane Orientation angles (deg):
(α=1.05, β=-36.82, γ=-9.52)

d2Target: 2.01 meters

Figure 4.17. Frame used in performance analysis and outcomes of its TRIP parsing.

Furthermore, using the UNIX `prof` tool, profiles of the TRIP parsing process were extracted, in order to determine the computational cost incurred by each of the different stages in which the TRIP parsing process is divided. Figure 4.18 shows the times in milliseconds spent in each of the stages when processing the mentioned 640x480 image. Likewise, the percentage of the total processing time spent in each stage is shown. Notably, the *adaptive thresholding* stage is by far the most computationally demanding. Nevertheless, the application of this stage is paramount for the correct operation of TRIP in environments where the ambient lighting is not controlled. Significantly, the overhead introduced by the `POSE_FROM_TRIPTAG` method is very small, which demonstrates that this method not only provides very accurate location information, but is also very computationally efficient.

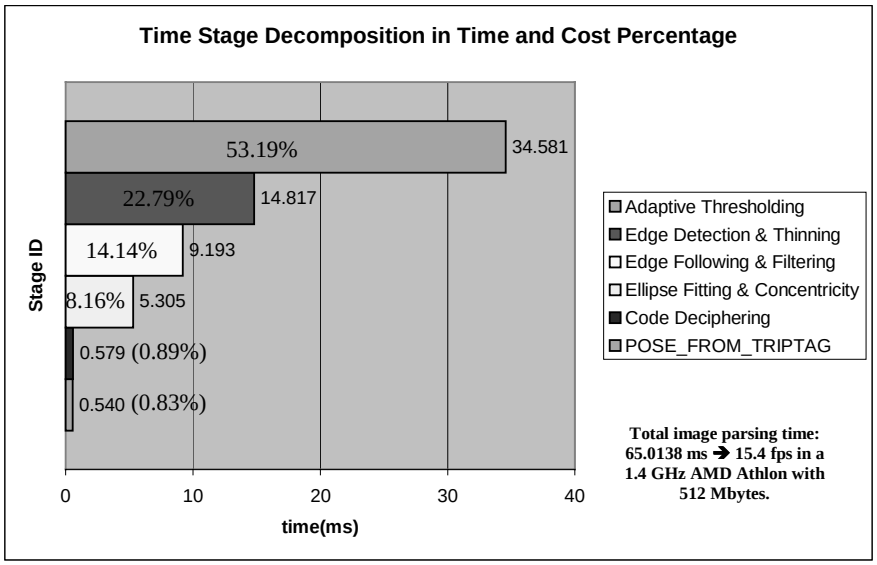


Figure 4.18. Performance results.

4.6 Conclusion

This chapter has shown how an accurate and high-performing 3D location sensor can be developed using the combination of video cameras and specially designed printable 2D circular tags. The `POSE_FROM_TRIPTAG` method is able to return a tagged-object location and orientation, with respect to the viewing camera, with a maximum error rate of 3% and 2%, respectively. This pose extraction method is valid for any marker whose geometric model provides a circular border of known radius and a point outside the circle lying on the *X*-axis of the marker centred coordinate system. TRIP offers a 3D coordinate location accuracy which compares favourably to the best existing location sensors, reviewed in chapter 2, whilst having fewer hardware requirements and presenting a more cost-efficient solution.

Nowadays, many computer users already own a very powerful PC as well as a printer and a low-cost web-cam (around $50). With just these tools, and the download of the TRIP software, a location system for everyday use becomes a reality. An average office room could be reasonably covered with just four or five web-cams. Provided the line-of-sight constraint of TRIP is upheld, useful sentient computing systems can be deployed rapidly and at low cost. The following chapters expand on the architectural support for TRIP (and other sensing technologies) and the applications produced using this 3D location sensor.

Chapter 5

Sentient Programming Abstractions

Software development for sentient systems is usually a rather laborious task, since it encompasses the cooperation of several distributed and heterogeneous elements, such as a network of sensors, a database, a location server, and the effectors undertaking the actions triggered by the sensor inputs. To tackle this, several research efforts have been conducted towards facilitating context-aware applications development. An extensive list of work in this area was reviewed in section 2.2. This work's approach to facilitating sentient applications development is described here and in the following two chapters.

Sentient Computing was defined in chapter 1 as the "ability of computing systems to *detect*, *interpret* and *respond* to the changing aspects of the user's local environment". Consequently, the development of sentient applications must address the following three aspects: (1) context capture, (2) context interpretation and modelling and (3) action triggering. In the previous two chapters, the context detection aspect of sentient systems has been addressed by proposing a low-cost and easily deployable vision-based 3D location sensor. Now, the context interpretation and modelling needs of sentient systems are tackled. Starting from the analysis of how the TRIP sensor can be modelled as a distributed sink of video frames and a source of contextual events, this chapter proposes a sensor-independent framework for modelling sentient applications. This framework defines three basic abstractions upon which sentient systems can be built, and an essential middleware service required for the discovery of available context sources and their capabilities. This chapter concludes identifying the need for two network-accessible infrastructure services to which application developers can outsource common functionality required by most sentient systems.

5.1 Using TRIP as a Source of Context Data

The identification and 3D location capabilities of TRIP, described in chapter 3 and 4 respectively, have been encapsulated into a video parsing C++ library that can be used by standalone C++ applications. This C++ library has also been wrapped through JNI [SUN97] into a Java package. Thus, users may also have access to TRIP from Java applications. Two versions of this software have been created which implement TRIP's default and adaptive behaviour (described in section 4.4), respectively. Although, TRIP is available for two of the most popular programming languages, it would still be desirable to make the TRIP technology platform- and programming language- agnostic. Moreover, taking into account TRIP's relatively high processing demands, it would be convenient to run a TRIP parser anywhere and independently of where frame sources are located. A frame source is an object supplying frames captured from a camera to interested parties. Therefore, there is a need to make the TRIP system distributed (or network-location independent). TRIP should be able to accept video frames from distributed frame sources and to export the video parsing results to interested parties in a convenient format.

The following three sections explain the model chosen to export the contextual information provided by TRIP to interested distributed applications. First, the middleware infrastructure used to make TRIP distributed, namely CORBA, is described. Second, the CORBA-based implementation of the TRIP's location sensor is explained. Finally, the video frame flow control requirements of the distributed version of TRIP are analysed and a solution is provided.

5.2 OMG CORBA

There are several object-oriented frameworks available for facilitating distributed applications development. The role of these frameworks is to offer a high-level, platform independent programming model to users, and to mask out problems of distribution. The most well-known and mature are: DCOM [Microsoft96], Java RMI [SUN99] and CORBA [OMG01]. More recently, some higher-level component- rather than object-oriented frameworks, such as the Java 2 Platform, Enterprise Edition (J2EE) [SUN01] or MicroSoft's .NET [Microsoft01], have also arisen. Such platforms aim to provide

underlying support for the third party development, composition and subsequent deployment of components, and also typically ease the task of the management of non-functional properties of applications (*e.g.* security or transactions).

Currently, these latter component-based frameworks are moving towards the popular topic of ‘web services', *i.e.* accessing distributed services by means of Internet standard technologies. Some examples of these standards are: (1) the Web Services Description Language (WSDL) [W3C01] used by providers to define a web service, (2) Universal Description, Discovery, and Integration (UDDI) registries where the providers register services or (3) the Simple Object Access Protocol (SOAP) [W3C00] that defines an XML format for implementing remote method invocations on web services over transport protocols such as HTTP. Unfortunately, web services frameworks claim to trade-off interoperability and more widespread access through commonly accepted Internet protocols, in exchange of worse request transmission efficiency and distributed programming flexibility.

For this work, CORBA has been chosen since it is a well-known standard and the only distributed systems enabling middleware that allows the efficient communication among multi-platform multi-lingual systems. CORBA provides the maximum flexibility for the development of distributed systems. Moreover, it is a mature technology with good documentation and freely available implementations. The following sections provide an overview of some CORBA fundamentals and a description of the CORBA Notification Service used in this work. Readers familiar with CORBA may skip these sections and proceed to section 5.3.

5.2.1 CORBA Fundamentals

The CORBA (Common Object Request Broker Architecture) [OMG01] standard, specified by the Object Management Group (OMG), defines a framework for developing object-oriented distributed applications. The core of the CORBA architecture is the Object Request Broker (ORB) that acts as the object bus over which objects transparently interact with other objects locally or remotely located. The ORB intercepts a client call and is responsible for finding an object that can implement the request, passing it the parameters, invoking its method, and returning the results. The client does not have to be

aware of where the object is located, its implementation language, its operating system, or any other system aspects that are not part of an object's interface. These heterogeneity features make CORBA ideal for the implementation of multi-purpose platform-independent components. A brief description of some essential CORBA concepts follows to familiarise the reader with this technology.

5.2.1.1 IDL

The Interface Definition Language (IDL) is a C++-like declarative language allowing the developer to define interfaces representing the methods and attributes associated with a CORBA object. IDL provides a set of standard built-in types and primitives to construct user-defined types.

5.2.1.2 GIOP and IORs

Interoperability in CORBA is addressed by the use of GIOP (General Inter-ORB Protocol) and standardized object references, called IORs (Interoperable Object References). GIOP is an abstract protocol that specifies transfer syntax and a standard set of message formats to allow independently developed ORBs to communicate over any connection-oriented transport. GIOP defines eight message types, from which *Request* and *Reply* are the workhorses that implement the basic RPC (Remote Procedure Call) mechanism. IIOP (Internet Inter-ORB Protocol) specifies how GIOP is implemented over TCP/IP.

IORs are opaque to client applications, but they contain the information ORBs need to establish communication between clients and target objects. Among other data, an IOR contains the *type name* of the most derived type of an object, the *protocol* (*e.g.* IIOP) and *address* details (*e.g.* IP address and TCP port number), and an *object key* identifying which particular object is pointed to by the IOR.

5.2.1.3 Static vs. Dynamic Method Invocation

The CORBA core provides for both static and dynamic access to objects. With static invocation, both the client and server have compile-time knowledge of all IDL types and interfaces they are using. This allows compile-time type checking, and improves

implementation efficiency. Use of static invocations involves passing object interface definitions through an IDL compiler, resulting in a set of language skeletons and stubs to be used by the server-side and client-side implementation code, respectively. The CORBA language mappings specify the interfaces of the generated stubs and skeletons from IDL into a specific programming language, *e.g.* C++ or Java. Optionally, the IDL compiler may store the generated type information for each method in an Interface Repository (IR). A client can query the interface repository to get run-time information about a particular interface and then use that information to create and invoke a method on the object dynamically through the Dynamic Invocation Interface (DII). Similarly, on the server side, the Dynamic Skeleton Interface (DSI) allows a server to respond to an operation invocation on an object for which it has no-compile time knowledge.

5.2.1.4 CORBA Object, Servants and Object Adapters

A *CORBA object* is a virtual entity, defined by an IDL interface, capable of being located by an ORB and having client requests delivered to it. A *servant* is a programming language entity that exists in the context of a server and provides the implementation for a CORBA object. An *Object Adapter* is the component responsible for adapting the programming language dependant servant to a CORBA object. The ORB and the Object Adapter cooperate to transparently locate and invoke the proper servants given the addressing information stored in CORBA object references. The standard object adapter, termed the Portable Object Adapter (POA), guarantees portability of servant code between different ORB products.

5.2.1.5 CORBA Services

Object services comprise of a set of IDL interfaces and semantic definitions, which provide common facilities for application and system developers. These services are collectively known as the Common Object Service Specification [OMG01c]. Some of the most commonly used CORBA services are: the (1) Naming Service, which maps an object name to an object reference; the (2) Trading Service, which provides a yellow pages-like service to lookup object references based on constraints, and (3) the Notification Service, which provides a mechanism for the asynchronous distributions of events among CORBA object peers. This last service is extensively used in this dissertation and is described in the following section.

5.2.2 CORBA Notification Service

The CORBA Notification Service [OMG00b] decouples the communication between CORBA objects and permits the interchange of events in an *asynchronous* manner through Notification Channels. A *Notification Channel* is both a supplier and a consumer of events. This object allows multiple suppliers to communicate with multiple consumers asynchronously and without knowledge of each other. The Notification Channel is responsible for supplier and consumer registration, timely and reliable event delivery to registered consumers, and the handling of errors associated with unresponsive consumers. This service also provides support for Structured Events, event type discovery, event filtering, filter sharing between consumers, QoS properties, and an optional Event Type Repository, where event type descriptions are stored.

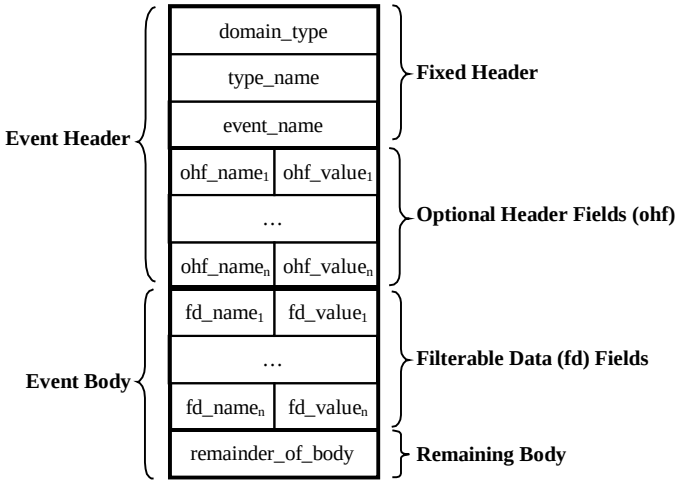


Figure 5.1. Structured Event format.

The Notification Service provides two models of event delivery: the *push* and the *pull* models. With a *push* model, suppliers take the initiative and push events to the Notification Channel that then subsequently pushes them to consumers. For the *pull* model, the actions that cause event flow occur in the opposite direction. Notification Channels not only support *push* and *pull* models for event delivery but also allow the models to be mixed.

5.2.2.1 CORBA Structured Event Format

Structured Events define a standard format for messages conveyed to a Notification Channel as shown in Figure 5.1. The `StructuredEvent` IDL structure consists of a header, a filterable body, and a remainder of body. The header comprises an event type, an event name, and an optional variable part that may contain QoS-specific attributes, timestamps, access rights and so on. Event types contain a domain name (D) and a type name (T). The domain name is used as a namespace in C++. The filterable body part contains attribute-value pairs upon which event-filtering constraints can be applied, while the remainder of the body is an opaque value.

In the context of this work it is assumed that every Structured Event supplied by an event source contains, in the filterable body, two event-independent attributes, namely `sourceID` and `timestamp`. The `sourceID` depicts a key representing the issuer of the event. The `timestamp` denotes when the event was issued. This latter field allows for temporal comparison of events and is an important requirement as the delivery of events within a distributed system is subject to delay. The clock synchronisation of every machine in a network hosting an event source or sink is necessary in order to establish comparisons between timestamps. For this, the Network Time Protocol (NTP) [Mills92] can be used.

5.3 TRIP: a CORBA-based Distributed Location Sensor

An event-based distributed architecture has been devised around TRIP in order to export the sensor data provided by this technology to interested applications. The functionality of the TRIP system, *i.e.* its target recognition and pose extraction, has been encapsulated within a CORBA object named *TRIPparser*. This component offers a UNIX `pipe`-like interface that enables applications to connect distributed *frame source* components, supplying images from cameras spread throughout the environment, to TRIPparsers. One-to-one pipes between a frame source and a TRIPparser are established. A frame source may pass its frames to several video sinks, but a TRIPparser will receive frames from only one source. This is necessary because a TRIPparser object needs to keep specific information associated with the camera from which it parses frames, *e.g.* intrinsic parameters or supported frame capture formats of the camera. Frame sources *push* image

frames into TRIPparsers. TRIP's image parsing results are, by default, *asynchronously* communicated in an event-form to a CORBA Notification Service's Channel [OMG00b] associated with each TRIPparser. This interleaved event communication component decouples the frame processing work from the result reporting duty. Thus, TRIP can concentrate on the CPU intensive image parsing process whereas the channel deals with the dissemination of the generated contextual data.

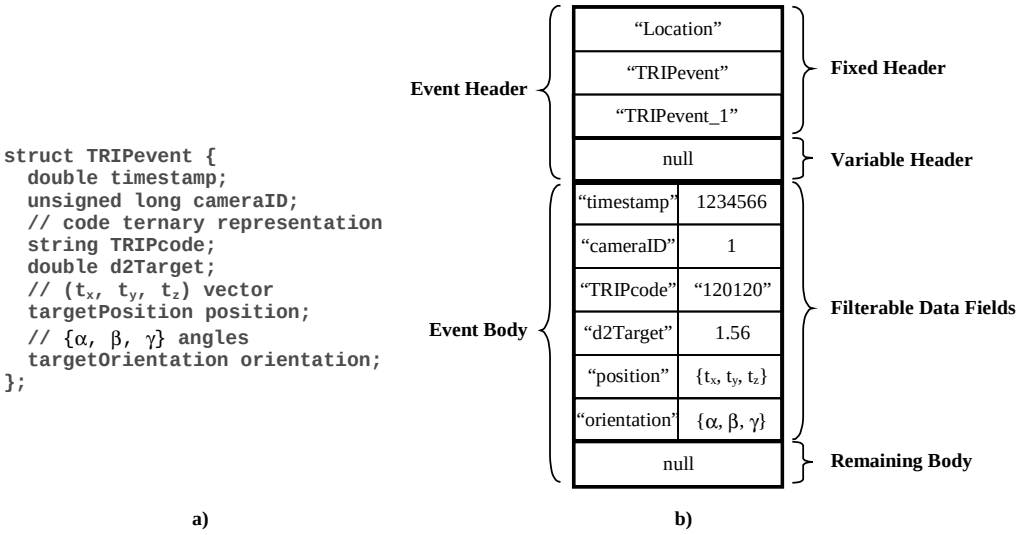


Figure 5.2. `TRIPevent` IDL structure (a) and TRIP Structured Event instance (b).

A TRIPparser acts as an event supplier of `TRIPevents`, represented in IDL in Figure 5.2a. These events are mapped into the `StructuredEvent` message type supported by the CORBA Notification Service. In fact, the TRIPparser generates a sequence of Structured Events, or *Event Batch*, corresponding to the TRIP sightings encountered within an image. The TRIPevent batch is transmitted by a single remote invocation to the Notification Channel. In the body of each Structured Event the contents of a `TRIPevent` are mapped into a set of name-value pairs to which filtering operations can be applied (see Figure 5.2b). Parties interested in a given TRIPparser's raw location data subscribe to its channel by passing a set of constraints over those name-value pairs, which are expressed in the Extended Trader Constraint Language [OMG00a]. The Notification Channel performs event filtering and communication on behalf of its representing TRIPparser. Hierarchical interconnections of TRIPparsers' Notification Channels can be created in order to ensure the efficient and scalable dissemination of TRIP generated data.

For example, TRIPparsers running in hosts within a room could push their events to the same notification channel, and similar notification channels for other rooms could be federated, in order to make available the whereabouts of TRIPtag wearers within a building.

In addition, a TRIPparser provides two synchronous invocation methods (`parseFrame` and `parseFrameRegion`) by which the TRIPparser pulls a frame or frame region, respectively, from the source passed as a parameter and returns the inferred location data. Hence, applications can interact with a TRIPparser in either a synchronous or asynchronous form.

5.4 Video Frame Flow Control in TRIP

TRIP forms part of a producer/consumer system, where *flow control* and the distinction between *push* and *pull* modes of operation are key issues. In a push system the source produces data and transmits it to the sink. The initiative is taken by the source and transmission is only slowed down when the consumer is slow. A pull system responds to the opposite behaviour, where the sink requests data. The following section states the requirements for a suitable frame flow control protocol for TRIP. After this, the *push model with token feedback* mechanism devised to address these requirements is described.

5.4.1 Requirements for a Frame Flow Control Protocol

There are three aspects to consider in the interaction between distributed frame sources and TRIPparser components:

1. Allow connections between arbitrary frame sources and TRIPparsers where the relative speeds of producer and consumer are *a priori* unknown and may differ by orders of magnitude.

2. Each source may have multiple sinks, and these will be of different speeds. A frame source may be feeding a viewing window which can be updated several times a second, as well as a TRIPparser which may cope with lesser number of frames per second.

3. The currency of a frame received by a sink is important. The TRIP system is supposed to respond speedily to real-world events. Frames cannot be stored in buffers for extended periods of time waiting for a slow component – a characteristic of many flow control systems. On the other hand, TRIP is not concerned with jitter-free frame rates. There are three reasons for this: (a) frames are just input data into TRIP that is never visualised, (b) TRIPparsers may be running on computers with heavy load, therefore presenting a bad frame-parsing rate and, (c) frames are timestamped.

5.4.2 A Token-based Flow Control Mechanism

Other researchers have tried to address a similar frame flow control problem. Notably, Stafford-Fraser [Stafford-Fraser+96] uses a combination of 'pushing' by the source and 'pulling' by the sinks. The sources generate frames at their own rate, uninterrupted by the sinks. A sink requests an up-to-date frame when it requires one and waits if it is not yet available. The sources, however, notice if the frames are not being consumed as fast as they are being produced, and slow themselves down to allow other components more processing time. They also notice if the sinks are waiting too long for frames, and attempt to speed themselves up. Unfortunately, the implementation of this solution does not address the distributed nature of both frame sources and video parsers in TRIP, forcing these components to reside within the same address space. Moreover, the implementation of this solution is dependent on a specific programming language, in this case Modula-3, whilst TRIP attempts to provide a platform- and language- agnostic solution.

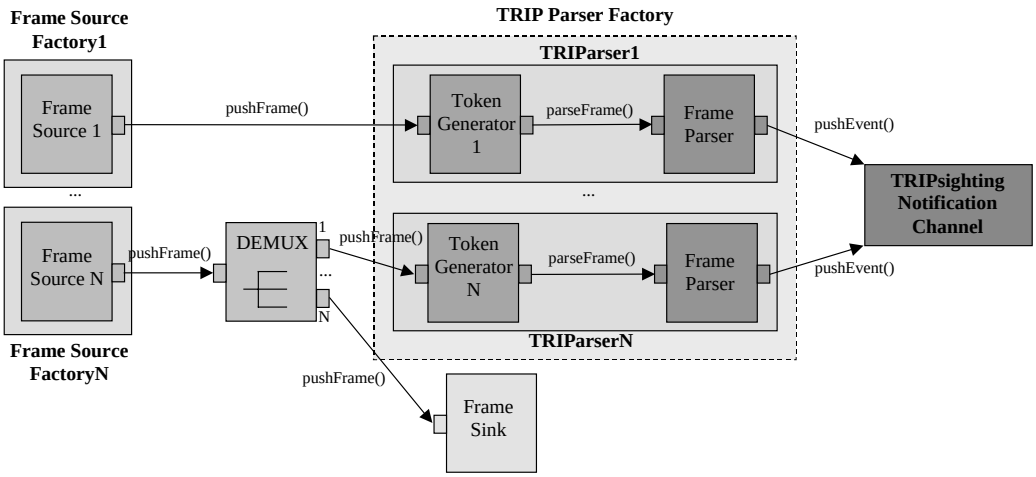


Figure 5.3. TRIP service architecture.

A token-based video frame flow control mechanism is used by TRIP. This mechanism aims to push frames to a parser only when it is expected that they are needed, and without explicit intervention from the TRIP sink. The information enabling a frame source to decide when it is suitable to push a new frame is given by a *token*, or special control data structure, fed back by the TRIPparser every time the analysis of an image is completed. This mechanism is aimed to make efficient the image flow control process and to integrate seamlessly with TRIP's adaptive operation (see section 4.4).

```
enum PixelFormat { P_GREY,   // 8-bit greyscale
                   P_RGB565, // 16-bit RGB-5-6-5
                   P_RGB555, // 16-bit RGB-5-5-5
                   P_RGB24,  // 24-bit RGB-8-8-8
                   P_RGB32   // 32-bit RGB-8-8-8
};

struct FrameFormat {
  unsigned short width;
  unsigned short height;
  PixelFormat pixFormat;
};

union RegOpt switch (boolean) {
case TRUE:
  Rectangle region;
};
```

```
struct Rectangle {
  unsigned short x0;
  unsigned short y0;
  unsigned short x1;
  unsigned short y1;
};

typedef sequence<octet> Data;
```

```
struct Frame {
  double timestamp;
  FrameFormat format;
  Data payload;
};
```

```
struct Token {
  FrameFormat format;
  RegOpt subwindow;
  long waitTimeout;
};
```

Figure 5.4. `Frame` and `Token` structures in IDL.

The architecture of the producer/consumer system designed for TRIP is shown in Figure 5.3. Frame source factories are containers of frame source components that push frames into TRIPparser components. TRIPparser components reside within the address space of TRIPparser factories. A TRIPparser is actually composed of a *Frame Parser* element which processes supplied video frames and a *Token Generator* element which, based on the results of a parsed image, creates a token containing a timeout indicating how long the source should wait before the next frame is pushed and in what format such frame should be supplied. The `Frame` and `Token` structures used as input and return parameters of the `pushFrame` and `pushFrameRegion` methods offered by the TRIPparser are shown in

Figure 5.4. Note that when the adaptive behaviour of TRIP is active, the frame source may be required to occasionally push only regions of new frames rather than the whole image. The `subwindow` field of the `Token` structure permits the frame source to realise if the TRIPparser expects an image or just a region of it. Upon an image region request, the frame source pushes the frame fragment required through the `pushFrameRegion` method. It is responsibility of the adaptive behaviour of TRIP to decide what region of an image should be pushed or even whether any images should be pushed at all for a while, *e.g.* TRIP is in the saving mode and decides to sleep for a short time.

Figure 5.5 shows the method invocations necessary for a controlling application to establish a frame communication pipe with `Token` feedback between a frame source and a TRIPparser component. The steps required are:

1. A controller application obtains an object reference to a frame source factory, *e.g.* from a Naming Service, and invokes its `createFrameSource` method. As result of this invocation, the factory object creates a frame source object in its address space and returns the new source's IOR to the controller application.

2. The controller application contacts a TRIPparser factory and issues a parser creation request in an analogous way to step 1. Using the returned TRIPparser's IOR, the controller issues a `connectFrameSource` invocation, passing as a parameter the IOR of the frame source with which the communication pipe should be established. As result of this, the TRIPparser requests to the frame source object for its camera configuration details, *i.e.* camera's intrinsic parameters and supported video formats. It then requests an image from the frame source of the maximum resolution supported and greyscale video depth. The TRIPparser processes the polled frame to establish an estimate of the time required to parse a frame. Based on the maximum frame size supported and the frame parsing time estimation, the TRIPparser creates an initial `Token` that is returned to the controller application.

3. The controller application invokes the `startPushingFrames` method on the frame source object passing the initial `Token` as an argument. As a result of this invocation, the frame source initiates two threads, in charge of pushing video frames

of the format and at the intervals required by the `Token`s retrofitted by TRIPparsers, and returns control to the application. The first thread immediately starts pushing the first frame, while the second thread waits for the timeout specified in the initial `Token` and then pushes the second frame. Every time a pushed frame processing terminates, the TRIPparser returns a new `Token` indicating how long a thread should wait before pushing a new frame (usually the processing duration of the previous frame) and the format in which this frame is required. Based on the `Token`'s information, the frame source dynamically decides when it is suitable to grab a new frame and push it to the parser.

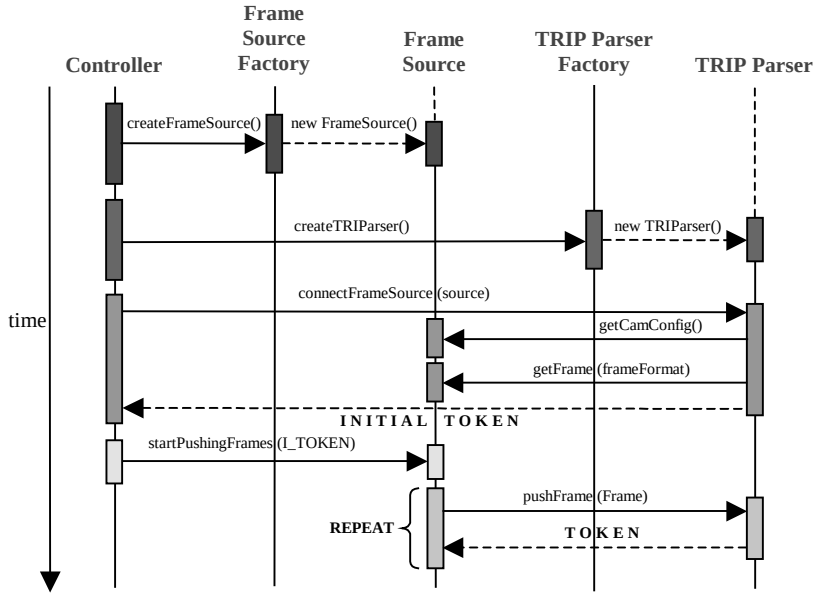


Figure 5.5. TRIP service interaction diagram.

The goal of this *frame pushing with token feedback* flow control mechanism is to make sure a new frame is available for a TRIP sink as soon as this has finished processing the previous frame. The frame source's pushing threads will normally sleep for the period required to process the last frame from which results were reported, thus guaranteeing the frame supplied is the most up-to-date possible. The net result of applying this frame flow control mechanism to TRIP is that the network transmission cost of a frame is practically negligible. This assertion holds as long as the time taken to process a frame is longer than the time employed to transmit a frame, as this protocol uses double frame buffering. In a conventional 100 Mbps Ethernet LAN, a maximum of 42 640x480 pixel images can be transmitted, whereas a TRIPparser will require at most 20 fps.

5.5 The TRIP Directory Service

A TRIP Directory Server (TDS) has been created with the purpose of regulating the TRIPcode granting process and establishing mappings between real-world objects and TRIPcodes. This component guarantees the efficient utilisation of TRIPcodes' address space and their classification into categories with a common ternary prefix, used by consumers in event filter registration. The TRIPcode allocation and prefix assignation process is fully described in Appendix C. The TDS offers CORBA interfaces for the creation, modification, deletion and retrieval of both TRIPcodes and their categories. On category creation, a data schema defining the types of the set of name-value pairs associated with TRIPcodes belonging to that category is required.

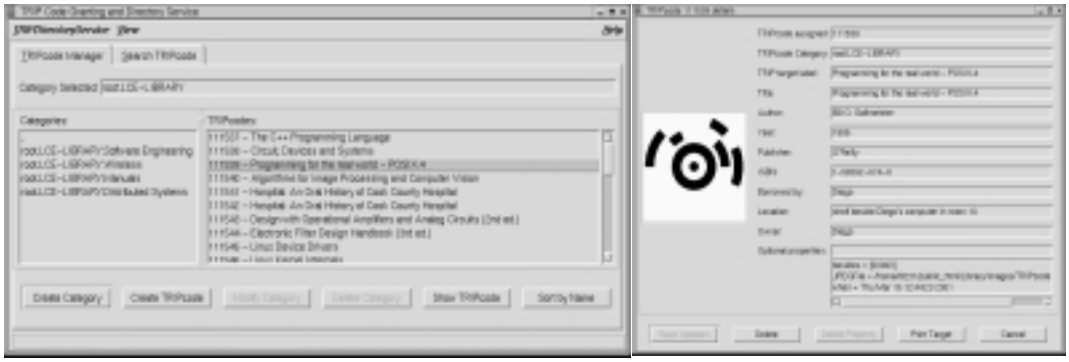


Figure 5.6. TRIP Directory Server's front-end screenshot.

The TDS also provides a notification mechanism suitable for applications that only wish to query this component during their bootstrap stage and still be aware of modifications to their TRIPcode's categories of interest. TRIPcode creation, modification and deletion events are conveyed to the TDS's associated notification channel. Figure 5.6 shows a screenshot of the TDS's GUI front-end.

5.6 The Sentient Information Framework

The event-based context distribution model applied to the TRIP sensor could easily be applied to model the behaviour of other context sources. Additionally, it would be interesting to extend this model to address not only context capture and dissemination but

also context interpretation. Notably, applications are often uninterested in the raw data coming from a sensor system, *e.g.* "TRIPcode 1,944 spotted by camera 1", but on the interpretation and abstraction of such information, *e.g.* "user Diego has been seen in room 1". All these factors have motivated the design of the *Sentient Information Framework* (SIF). The founding goal of SIF is to produce a reusable and customisable set of building blocks for context sensing and interpretation. This framework focuses not only on providing suitable high-level abstractions on which sentient application programmers base their designs, as provided by other previous research [Salber+99][Hong+01] [Brown+97], but also pays special attention to efficient sensor data dissemination.

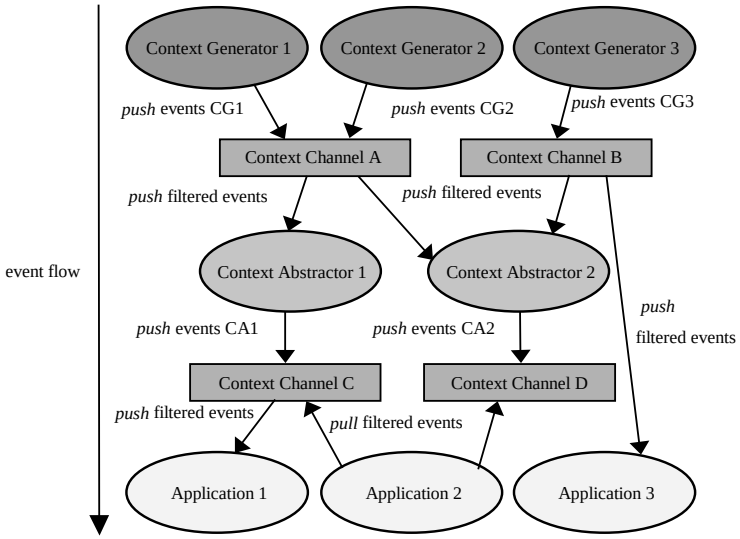


Figure 5.7. SIF architecture.

SIF defines an application construction model to streamline sentient systems development. SIF isolates context capture and abstraction from application operation and, at the same time, provides efficient mechanisms for context communication. Its main function is to transform context information into the format demanded by applications, rather than forcing them to deal with the low-level intricacies of sensor access and data interpretation. SIF-enabled applications simply subscribe for the high-level contextual event notifications that can directly drive their operation. For example, a door access control system would conventionally have to interact directly with the underlying identification technology, *e.g.* TRIP, and interpret the sentient data provided, to determine when an authorised user is detected. Under SIF, however, this application would simply

register for an "authorised user's presence" event, since SIF undertakes the sentient data manipulation and interpretation on its behalf.

The architecture of SIF (see Figure 5.7) consists of a group of co-operating distributed software components that use events as a uniform way of informing each other of context changes. SIF defines three main component *abstractions* upon which the design of sentient applications may be based:

1. *Context Channel* (CC) objects sit between other SIF components to de-couple their operation from event filtering and communication duties. Through them, multiple suppliers can communicate transparently and asynchronously with multiple consumers without the components knowing about each other. They are shared by co-operating components that either generate or consume events of the same type. Upon event arrival they undertake event filtering and transmit only those events for which components have registered interest. Context Channels actually correspond to CORBA Notification Channels.

2. *Context Generators* (CG) encapsulate a single context source or a set of related ones and the software that acquires raw context information from them. Every Context Generator communicates the sentient data captured to its associated Context Channels. Context Generators may be based on physical sensors such as TRIP, GPS, or a microphone. They can also represent a software object, such as the TRIP Directory Server, that communicates notifications upon a state change.

3. *Context Abstractors* (CA) achieve the separation of concerns between context sensing and application semantics. They consume raw sentient data provided by Context Generators (*e.g.* "TRIPcode 1,944 spotted"), interpret its contents (*e.g.* "user Diego spotted") and augment it (*e.g.* "Diego's login is dipina"), producing enhanced contextual events that directly drive applications. They may also provide event aggregation, *i.e.* a set of primitive events coming from other Context Abstractors or Context Generators is summarised by a single new event.

Both Context Abstractors and Context Generators are designed for asynchronous and decoupled interaction with other SIF components by means of their associated Context

Channels. Nevertheless, they also provide synchronous CORBA interfaces to enable applications to query the latest context they have monitored. Querying a CA or CG through the synchronous interfaces is appropriate for one-time context needs or when an application is first started. The asynchronous *publish/subscribe/match* event mechanism provided by the SIF framework is appropriate for repetitive context needs.

An interesting consequence of having implemented Context Channels using CORBA Notification Channels is that event suppliers offer an event *quench* mechanism. Through this mechanism, event suppliers can dynamically realise whether there are consumers interested in the events they are producing or not. This facility permits context sources to intelligently stop or resume event communication, hence making an efficient use of network bandwidth.

Context Abstractor components are the core components of the SIF architecture. They undertake the separation of context capture and interpretation from context use by transforming context information to the end application needs. Their behaviour responds to an *Event-Condition-Action* (ECA) model. They monitor asynchronous event communication, apply conditional statements to them and, whenever a condition is fulfilled, generate an action (in this case, produce a higher level event). Between a Context Generator and a final application, an event can flow through N-tiers of Context Abstractor components. For example, a person's TRIPtag sighting event, "code 1,944 seen by camera 30", could be transformed by a *Location Service Context Abstractor* into "Diego seen at entrance corridor's front-door region". In turn, a *Laboratory Access Context Abstractor* could map this latter event into an "authorised user's presence" event. This enhanced contextual event would finally reach the application in charge of performing the actual action, *i.e.* the *door access control system*.

The main benefits of adopting SIF for sentient application construction are its run-time *reusability* and *extensibility* features. Context Channels are shared by context sources (Context Generators and Context Abstractors), which produce events with previously agreed semantics, and by consumers (Context Abstractors and applications) interested in that contextual data. New Context Abstractors can be interposed in between other Context Abstractors or Context Generators and applications in order to adapt and extend the contextual information flowing through SIF to the end application needs. The context

abstraction mechanism proposed permits an application to receive context from an alternative sensor system to the one initially used (provided the new sensor supplies events with the same semantics as the previous one), without requiring any change in the application code. This property can be useful for Sensor Fusion [Brooks+98] as described in section 5.7. The final aim with SIF is to create a catalogue of reusable and extensible components. Some example SIF components are the `TRIP`, `ActiveBadge` and `SoundLevel` Context Generators and the `LocationService`, `Meeting_ON` and `PeopleTogether` Context Abstractors.

5.6.1 The Context Trader

A middleware service, named *Context Trader*, has been designed and prototyped to assist SIF components and applications in the discovery of other SIF components and their capabilities. Its operation subsumes and extends the functionality of a conventional Trading Service [OMG00b]. When a SIF component is started, it exports its object reference, the object reference of its associated notification channel, and the metadata associated with the types of events and the values of the event attributes it can provide. In addition, the component also reports metadata about the set of services it can provide, *e.g.* frame grabbing capabilities, to the Context Trader. Applications may later request, through constraint expressions, the IORs of SIF components providing a given type of contextual event, or whose events contain the set of attribute name-values specified. Furthermore, applications may register with the Context Trader to be notified when new context sources providing a certain kind of events are added, or when existing ones are deregistered. This permits sentient applications to adapt to changes in the SIF framework.

5.6.2 SIF Implementation

The omniNotify [AT&T01a] implementation of the CORBA Notification Service is used by SIF. C++ and Java libraries that ease the implementation of components following the SIF abstractions have been produced. These libraries undertake the end-to-end communication of information in a transparent manner for applications, i.e. isolating the use of CORBA middleware from applications, and enable their interaction with the Context Trader.

5.7 Sensor Fusion through Context Abstractors

In order to validate the SIF sentient application construction model, a prototyped *Location Service Context Abstractor* is described. The mission of this component is to fuse the raw contextual information coming from two different location systems (Active Badge and TRIP systems), and transform such data into presence and movement events directly usable by location-aware applications. Note that this Location Service CA is interested only in sighting events associated with personnel and not with other type of tagable assets such as equipment or books. For this reason, the Context Abstractor registers event filters (expressed in the Extended Trader Constraint Language) with the Context Channels of the location systems in order to limit the events received to be personnel-related.

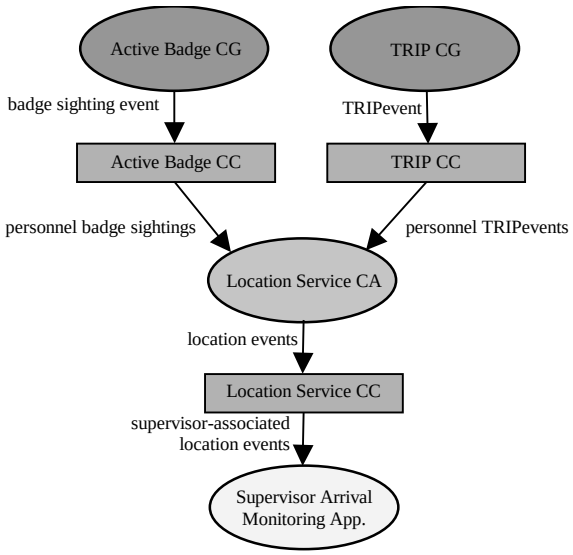


Figure 5.8. Location Service CA.

The Location Service CA transforms raw event notifications from the TRIP and Active Badge systems into movement or presence events expressed in terms of regions. For example, “TRIPcode 1,944 has been seen by camera N at a distance D with an orientation O”, would be transformed into a location event directly usable by an application, *e.g.* a “Diego seen at Room 1’s camera N region” presence event or “Diego moved from Room X’s camera M region to Room 1’s camera N region” movement event. Likewise, the Location Service CA maps raw Active Badge sightings such as “Badge ID 0-0-0-0-13-a7

spotted by infrared sensor CUED-WT#0.8" into the corresponding location event. Logically, in this case, given the lower location granularity of the Active Badge system the regions reported will be the room containers, *e.g.* "Room X".

The Location Service CA isolates location-aware applications from the underlying sensing technologies. These applications are only interested in being notified about personnel location events, regardless of whether the users of interest are using an Active Badge or a TRIPtag. Thus, applications simply register with the Location Service CA's Context Channel, passing event filter specifications with constraints over the events that want to be received. For example, a student may wish to develop a supervisor arrival monitoring application that notifies him when his supervisor enters the laboratory. For this purpose, the student would write an event filter constraint expression limiting the presence or movement events received to be associated with his supervisor. Figure 5.8 shows schematically how this application would be integrated within the SIF architecture.

5.8 Related Work to SIF

The Sentient Information Framework (SIF) presents some resemblance to the Situated Computing Service (SitComp) [Hull+97] and Context Toolkit [Dey+00] projects described in section 2.2. SIF borrows from SitComp its approach to abstract the context captured to the needs of final applications. Nevertheless, SIF is fully distributed whereas SitComp enforces the context capture and interpretation logic to be embedded within a single server process. This makes SitComp difficult to extend when new context wants to be captured or different context interpretations want to be performed. SIF's *Context Generator* component is very similar to the Context Toolkit's *context widget* abstraction. Likewise, SIF's *Context Abstractors* can serve as both the *context aggregators* and *interpreters* abstractions of the Context Toolkit system. SIF distinguishes itself from the Context Toolkit in its use of a special abstraction, namely *Context Channel*. This abstraction decouples the context filtering and communication duties from the components' context capture or interpretation duties. Consequently, the integration within SIF of new components is simpler than in the more tightly coupled Context Toolkit architecture. In addition, CORBA-based Context Channels are devised to carry out the dissemination of context information in a very efficient manner, unlike the HTTP-based

communication mechanism used by the Context Toolkit. SIF and the Context Toolkit share the common goal of providing a set of re-usable components that can be combined easily to build a sentient application.

5.9 Middleware Services for Sentient Computing

The main aim of this work is to make Sentient Computing more accessible to everyone and everywhere. In order to fulfil this purpose, a new cost-effective and easily deployable location sensor technology, TRIP, and a sentient application construction model, SIF, for the efficient manipulation and dissemination of both TRIP and other sensor technologies' sensor data have been illustrated. Therefore, both the *context capture* and *context interpretation* aspects of sentient systems development have been tackled. Nevertheless, further support for context interpretation and abstraction and for the third aspect in the development of sentient systems, *i.e. action triggering*, is needed.

The development of several Context Abstractors has highlighted that there exists a lot of inconvenient code duplication due to common behaviour patterns. Every Context Abstractor undertakes a very similar task. Firstly, they register with a set of Context Generators' Context Channels passing some event filter constraints. Secondly, they correlate the different events arriving. Finally, they generate a higher-level event which can be directly consumed by another SIF component or an application. Therefore, it is inefficient to force each CA to undertake this task separately. It would be convenient to have a middleware service to which the sentient applications could delegate their event correlation duties. That way the development of Context Abstractors would be simplified and limited to providing access to databases and other repositories with the information necessary to interpret event fields. For example, a CA in charge of notifying users about the closest policeman to them may have to filter location sightings of personnel to verify whether a user's ID corresponds to a policeman. Ideally, sentient application development would be simplified to the extent that the programmer only had to register an event composition specification with the network-based event correlation service, and then implement a handler for the notification triggered by the service when the situation requested is matched.

Furthermore, an infrastructure that assists a sentient application to trigger the expected action, *i.e.* activate, deactivate or migrate a user-bound software service, upon the arrival of a high level event (*e.g.* "Diego enters his office"), is desirable. Experimentation with sentient application development has shown a need for a middleware infrastructure to streamline user-associated services' lifecycle control. Otherwise, every time a new sentient application is developed, the distributed components involved in its operation must be manually started. The lack of this middleware makes sentient application deployment very cumbersome and resource wasteful. Moreover, user-related services are often bound to user location, *e.g.* the activation of an MP3 player when a user presence in a room is detected, must take place in one of the PCs of that room. Therefore, it is not only desirable to control the lifetime of user-bound services but also their location in the network.

As a result of the above observations, two middleware services have been developed to assist sentient application developers in two vital areas: (1) rule-based monitoring of contextual situations and (2) user-bound service lifecycle and location control after abstracted event arrival. Chapters 6 and 7 will demonstrate how those two middleware services substantially alleviate the programming effort required of developers of sentient systems.

5.10 Conclusion

This chapter has shown an event-based model for exporting TRIP generated contextual data to interested applications. Frame flow control issues within the TRIP system have also been addressed by the design of a *frame push with token feedback* protocol. The event-based distribution model devised for TRIP has been generalised to other sensing technologies and to enable the key task, within sentient systems, of context interpretation and abstraction. This effort has resulted in the development of the Sentient Information Framework. This framework defines three component abstractions upon which sentient applications can be efficiently modelled, and a middleware service through which SIF components and their capabilities can be searched. Finally, experimentation in the development of sentient applications, by means of TRIP and SIF, has illustrated the need for the development of two essential middleware services that will help in: (1) the rule-

based context correlation duties and (2) the action triggering aspect of sentient systems. These middleware services are discussed in the following two chapters. They will complement the TRIP and SIF contributions in the achievement of this dissertation's goal, *i.e.* making viable a more widespread adoption of Sentient Computing in our working or living spaces.

Chapter 6

A Rule Paradigm for Sentient Computing

Sentient systems are reactive systems that perform actions in response to contextual events. They respond to the stimuli provided by sensors distributed through the environment by triggering actions that aim to satisfy the user's expectations based on their current context, *e.g.* their identity, location or current activity.

The development of even a conceptually simple sentient application usually involves the correlation of inputs provided from diverse context sources. Typically, a sentient application has to register with many distributed event sources and to monitor the occurrence of a pre-defined combination of events. Each asynchronous low-level occurrence supplied by a source is termed *primitive event*. The asynchronous occurrences triggered by the correlation of many primitive or composite events (that are monitored by sentient applications) are termed *composite events*.

Consider, for example, a sentient application that wants to switch on the heating system in a room when at least one person is there and the current temperature is below a given threshold. The application would have to register with a location system event source to receive personnel presence events in that room and with a temperature event source to be notified when the temperature goes below a threshold. When the composite event is matched (at least one user is in the room and the temperature is below the threshold), the application will initiate the heating system.

As observed by this example, the *modus operandi* of sentient applications, and reactive applications in general, follows a common pattern, *i.e.* they wait until a pre-defined *situation* (a composite event pattern) is matched to trigger an *action*. In other words, sentient applications respond to an *Event-Condition-Action* model. Consequently, it is

ineffective to force each sentient application to handle this behaviour separately. A generic middleware capable of undertaking the composite event monitoring process, *i.e.* the correlation of multiple primitive or composite events, on behalf of sentient applications is hence desirable. This chapter describes the implementation of an Event-Condition-Action Rule Matching Service (ECA Service) addressing this issue. This service is based and extends the functionality of the CORBA Notification Service.

6.1 Limitations of the CORBA Notification Service

The main function of the CORBA Notification Service [OMG00b], discussed in section 5.2.2 and used by SIF, is to decouple the communication between CORBA objects by permitting their interchange of events in a *publish/subscribe/match asynchronous* form through Notification Channels.

The main limitation of the Notification Service is that it enables filtering only on primitive events and does not support compound event matching nor event aggregation. *Event Composition* is defined as the process by which the set of events defining a situation is notified to the client application. The application may then process each primitive event part of a composite event. *Event aggregation* is defined as the process by which a set of matching events is summarised by a single new event.

The proposed ECA Service addresses the event composition and aggregation limitations of the Notification Service. The initial format of events supported by its implementation is the Structured Event format (see section 5.2.2.1) defined by the OMG's Notification Service. However, the modular architecture devised for the ECA Service allows for the simple integration of other types of event notification services into this system.

6.2 CLIPS as a Tool for Event Composition

Declarative languages such as Prolog [Bratko00] or CLIPS (C Language Integrated Production System) [NASA99], used in the context of Logic and Expert Systems Programming are specialised in the description and analysis of relationships. Given a set of *facts* (true propositions about entities) and *rules* applied to them, an *inference engine*

built into the language can decide which rule to 'fire'. In order to solve a problem, programmers only need to specify a set of *situation-action* (or production) rules and facts. The *situation* (IF) component takes the form of predicates applied to values of attributes of a specified object. The *action* (THEN) component of the rule produces new knowledge or triggers an action. Surprisingly, very few researchers [Koschel+98][Stafford-Fraser96] have exploited the inherent benefits of these languages in the specification of real-time reactive systems' behaviour rules.

This work builds on the built-in pattern matching capabilities of CLIPS' inference engine to provide a middleware service that undertakes the common event composition and aggregation tasks required in the implementation of reactive systems. CLIPS is suitable for the following reasons:

1. *Rich expressive power*. CLIPS permits the specification of very complex relations of event patterns. It enables the use of a rich set of relational predicates (*e.g.* =, >) and connectives (*e.g.* AND, OR, NOT) over fact instances to build compound conditions.

2. *Performance*. CLIPS' built-in inference engine implements the RETE [Forgy82] algorithm, a very efficient mechanism to solve the difficult many-to-many pattern matching problem. RETE is very suitable for the event correlation task in mind. RETE minimises the number of tests required in the matching process; partial pattern matches are stored in a *rete* (latin for 'network'), where they do not have to be re-tested, and new objects arriving in the network are tested only where necessary.

3. *Integration/extensibility*. CLIPS is designed for full integration with languages such as C++ and Java. It can be used as a stand-alone tool, but it can also be called from a procedural language. Likewise, procedural code can be defined as external functions and called from CLIPS.

Despite its advantages, CLIPS is not a suitable programming language for the average programmer because:

1. *Unconventional syntax*. CLIPS syntax, based on LISP, differs significantly from the syntax of other more common procedural or object-oriented programming languages

such as Pascal, C++ or Java. For instance, the programmer is forced to use many parentheses in the code and to write expressions in reverse polish notation rather than in the conventional infix notation.

2. *Generality*. CLIPS is a language for the production of generic rule-based systems. A simpler language that provides high-level constructions for the more specialised domain of composite event monitoring is desirable.

The next section describes a rule-based event composition language – the ECA language. This language hides the inconvenient syntax of CLIPS (still keeping its rich expressive power) and adds suitable extensions for composite event management. The main reason for developing a new language has been to relieve the programmer from coding the cumbersome temporal constraints associated with event composition. The programmer simply specifies rules in the ECA language and the compiler of this language automatically generates the temporal constraints that otherwise would have to be specified by hand in CLIPS.

6.3 A Language for Specifying ECA Rules

Appendix D shows the grammar of the ECA language. The main component of this language is the `rule` construction. An ECA rule specification is composed of a set of patterns applied to the contents of events together with the actions that are triggered by successful matches:

```
<rule> ::= {<pattern_list> => <action_list>}
```

*Event template*s are the most common type of pattern within a rule. An event template contains the domain name and type name of the event and a set of name-value pairs corresponding to the attributes of the event that is used in the matching process. For instance, `Location$presence(user "Peter", room ?roomID)` matches the first event belonging to domain `Location` and of type `presence`, where the attribute `user` is equal to the constant "`Peter`" and the contents of the attribute `room` can have any value and are assigned to the variable `?roomID`. Attribute name identifiers may be separated by

dots to denote nested attributes, *e.g.* `location.Xpos`. '?' indicates a variable declaration. Optionally, event patterns may also be preceded by a variable. Variables representing events are used when the set of primitive events part of a composite event is communicated to a client as an *event batch*. Thus, the generic form of an event pattern is expressed as:

```
[<event_var>:]<domain>$<event_type>(<name_value_list>)
```

Simple event matching expressions can be combined to form compound matching expressions which describe a composite event using the operators `and`, `or`, `not`, `before`, and `after`. The semantics of the first two operators coincides with the semantics of their analogous operators in conventional programming languages. A `not` operator preceding an event template signifies that no occurrence of that event should take place in the period when the other primitive events part of a composite event occur. The `before` operator indicates a temporal sequence, *i.e.* events patterns situated on the left-hand-side (LHS) of this connective must temporally precede the event patterns on its right-hand-side (RHS). The `after` operator represents the opposite temporal behaviour.

A `test` expression can be applied to variables corresponding to the fields of event templates, to further constrain the set of possible matches. This expression must return a Boolean value. This construction is useful for expressing both intra- and inter- event constraints. Conventional relational and arithmetic operators are used to build an expression, *e.g.* +, *, < or =.

The action part of a rule is executed when the antecedent of the rule has successfully been matched. The supported action statements are: (1) variable assignment, (2) event notification and (3) command execution.

```
<action_stmt> ::= <VARIABLE> := <expr>; |
      notifyEvent(<event>); |
      fireAction(STRING,<params_list>);
```

An assignment action binds a variable to an expression. This is most often used to create the fields composing a new aggregated event or the parameters to be passed when firing

an action. An event notification action may be triggered when either the set of primitive events corresponding to a matching composite pattern is communicated as an event batch or when a new aggregated event is generated. The parameters passed within an aggregated event correspond either to constant values or to variables declared before the statement. The `eventBatch` statement must pass as parameters only variables that have previously been bound to event templates. The format of an event passed in a notification action is:

```
<event> ::= <domain>$<event_type>(<event_params_list>) |
            eventBatch(<event_var_list>)
```

Finally, an action may be fired in response to a matched situation. The action name may correspond to one of the common actions available with the implementation of the ECA Service, *e.g.* `sendEmail`, `playSound` or `text2Speech`, or correspond to the file path of an arbitrary script created by the application designer.

6.4 A Middleware Service for ECA Rule Matching

The main function of an instance of an ECA Rule Matching Service, or ECA Server, is to accept, match and manage Event-Condition-Action rule specifications on behalf of a client application. An ECA Server receives both ECA rule registrations and CORBA Structured Events. As a result of a situation match, an ECA Server triggers one the following three actions:

1. *Event Composition*: the ECA Server notifies a consumer with the batch of primitive event instances corresponding to a composite event pattern match.

2. *Event Aggregation*: the ECA Server notifies a consumer with a newly generated event summarising an event correlation.

3. *Action Execution*: the ECA Server executes the action triggered as result of a matching situation. This action may correspond to a set of common actions provided by the service, or an arbitrary script specified by the user.

The ECA Service assumes that clients registering ECA rules implement the Notification Service's `SequencePushConsumer` interface. This interface offers an operation that enables a consumer to receive a sequence of Structured Events (event batch) via a *push* notification model.

6.4.1 ECA Server Architecture

Figure 6.1 shows the building blocks of an ECA Server. The heart of an ECA Server is an embedded CLIPS inference engine. This inference engine determines when the LHS of user specified rules is matched by the current set of facts stored in the *knowledge base* of the engine. As a result of this process, the engine initiates the actions specified in the RHS of rules.

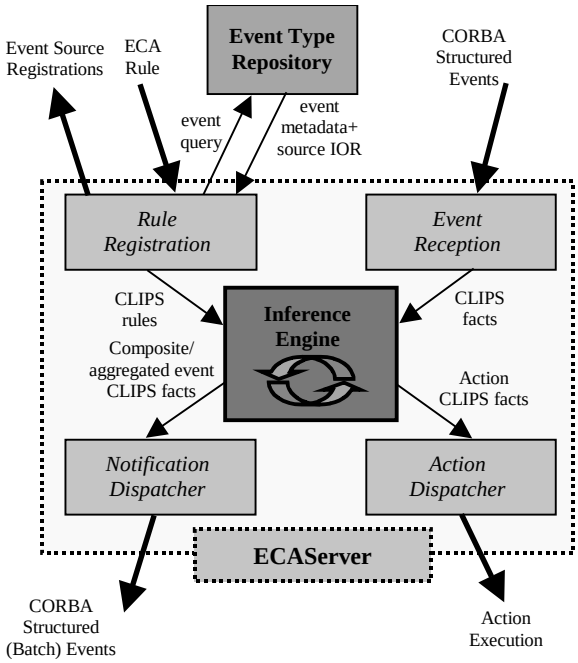


Figure 6.1. ECA Server architecture

The *event reception module* populates the knowledge base of the embedded CLIPS engine with facts representing the events coming from distributed event sources. This module understands events supplied by a given event notification platform, in our implementation OMG's Notification Service's Structured Events, and maps them into event representing facts that are understood by CLIPS. To map Structured Events to

CLIPS facts, the data type introspection capabilities of CORBA [OMG01] are used. These are the `TypeCode` and `DynAny` interfaces.

The *rule registration module* maps ECA rules specified by a client to CLIPS rules. To test the validity of the event templates passed within a rule specification, this module needs to query an external Event Type Repository (ETR)[17]. This component is populated with metadata describing the structure of all known event types that may be supplied to event source's Notification Channels. Event sources send the metadata of the events they generate and the object references of the Notification Channels where those events are supplied to the ETR. The rule registration module queries the ETR to retrieve this data for the event types expressed within a given ECA Rule. Then, it registers with the channels providing the event types required as an event consumer implementing the Notification Service's `StructuredPushConsumer` interface.

The *notification dispatcher* maps CLIPS facts representing aggregated events, supplied by the inference engine, into CORBA Structured Events. The CORBA introspection capabilities are used to generate (on-the-fly) aggregated events whose internal format is unknown at compile time. CLIPS facts representing a composite event match are mapped into a batch of CORBA Structured Events. The generated events are then communicated to the clients that registered the firing rules. This event notification capability coverts the ECA Service into a CORBA Notification Service-compliant source of Structured Events.

The *action dispatcher* processes facts representing actions sent by the engine and, as a consequence, either triggers some of its built-in common actions or initiates a user specified script.

The modular architecture of the ECA Server permits the replacement of the modules specific to the event format of a given middleware notification technology by others suited to another notification technology. Alternatively, *event adapters* could be interposed between event sources and the ECA Service, and *vice versa,* in order to translate a required event format to/from the canonical format chosen, *i.e.* the CORBA Structured Event format.

[17] The ETR and the Context Trader described in section 5.6.1 are, in fact, the same component.

6.4.2 Storage and Cleanup of Event-representing Facts

Events are discrete temporal occurrences. Some event sources produce events at regular intervals, others irregularly. Furthermore, the average frequency of event reporting differs from one event source to another. On the other hand, the CLIPS-based implementation of the ECA Service implies that facts representing Structured Events may be added indefinitely to the knowledge base of the embedded CLIPS inference engine. Therefore, it is necessary to store some of the events coming from event sources in the inference engine, so that the matching engine can use the last event reported by an infrequent source of events. Still, the number of events stored should be minimised in order to reduce the memory requirements of the engine.

A fact representing an action, added to the knowledge base of the engine as a result of the action component (RHS) of a rule, is retracted as soon as the action is executed. The approach chosen to tackle event-representing facts' garbage collection is to make one or more attributes of an event type *primary*, similar to the case where some attributes in a database relation are considered as *primary key*. Then, for each differing value of these primary attributes the most recent corresponding event is cached. For example, in the case of the `presence` event mentioned above, the attribute `user` would be considered primary so that the ECA Service would store only the last occurrence of a given user's `presence` event. In order to indicate that the ECA Service should match an event template with the last occurrence of an event rather than its default behaviour of waiting for a new instance of it to come, an event template expression in the ECA language must be preceded by the keyword `query`. For example, an infrequent source of events will be a login monitor notifying when a user logs into a machine. To build a composite event containing this event pattern the following expression could be defined: `query PCMonitor$logged_in(user ?userID, host ?hostID)`.

As part of the ECA-to-CLIPS rule-mapping process, the compiler will generate a rule for each event template to update instances of that event type based on the fields considered to be primary. For instance, for the case of the `presence` event, the rule will force the CLIPS engine to keep in its knowledge base only the most recent `presence` event associated with a user. The rule would read in CLIPS as shown in Figure 6.2.

```
(defrule updatePresenceEvent
 ?oldEvent<-(Location$presence((user ?userID)
                               (timestamp ?time0)))
 (Location$presence((user ?userID) (timestamp ?time1)))
 (test (> ?time1 ?time0))
=>
 (retract ?oldEvent))
```

Figure 6.2. Upgrading a `presence` event fact cached in the engine's knowledge base.

For the case of events representing the advent of a transient state (*e.g.* `door$opened` or `door$closed`), where two distinct event types may contradict one another, a different garbage collection scheme is necessary. The developer must register with the ETR, as part of the metadata of an event, the *dependencies* of that event with other types of events. For example, the programmer would specify the dependency between the `door$opened` and `door$closed` events. Thus, the ECA-to-CLIPS compiler, based on the dependencies and the primary fields of events, generates rules invalidating an event when a contradictory one occurs. Such a rule for the case of the `door$opened` event contradicted by a `door$closed` event would be:

```
(defrule invalidateDoorOpened
 ?oldEvent<-(door$opened((door ?doorID)
                         (timestamp ?time0)))
 (door$closed((door ?doorID) (timestamp ?time1)))
 (test (> ?time1 ?time0))
=>
 (retract ?oldEvent))
```

Figure 6.3. Invalidating a `door$opened` event with a more recent `door$closed` event.

6.4.3 Rules' Lifetime and Activation Time Span

Depending on the needs of the rule-registering client, a rule may be applicable within an ECA Server permanently, or may be fired just a fixed number of times. When a client registers an ECA rule, by invoking the `addECARule` remote method on the ECA Server, a parameter containing the number of times a rule should fire must be passed. When this parameter is zero, the associated rule will be active until the client explicitly deactivates the rule by invoking the method `removeECARule`. Otherwise the rule will be deleted following its last allowed firing.

On the other hand, the evaluation of a rule may need to be active up to a maximum period of time, *i.e.* a rule may need to ensure that correlated events occur within a given time span. The event detection time associated with an ECA rule is set by preceding a rule declaration with a `within` statement. For example the statement `within 5000` would mean that the events correlated in that rule should occur within a five second time period. Therefore, the syntax of an ECA rule is updated to the following expression, where `PERIOD` stands for the time interval, in milliseconds, in which a situation evaluation is meaningful.

```
<rule> ::= [within PERIOD] {<pattern_list> => <action_list>}
```

6.4.4 Extensions to the ECA Language

The current implementation of the ECA language does not support attributes representing a sequence, nor the index-based access to items in the sequence. The `in` operator for testing the containment of an item in a sequence and the square bracket operators (`[]`) to access items in a sequence would be interesting additions to the ECA language.

6.4.5 Implementation Details

A Java implementation of the ECA Service has been completed using the CORBA ORB included in Sun's JDK 1.4 [SUN02] distribution. A Jess (Java Expert System Shell) [Friedman-Hill01] inference engine, embedded in an ECA server, undertakes the rule-based reasoning. Jess is a Java clone of CLIPS that provides neat integration with Java. The JLex [Berk+00] lexical analyser and CUP [Hudson99] parser generator tools, Java versions of the well-known Lex and YACC tools, are used to implement the mapping from ECA rules into CLIPS rules. omniNotify [AT&T01a] was the CORBA Notification Service implementation used.

6.4.6 Performance of the ECA Service

An experiment was carried out to evaluate the event correlation performance achieved by the ECA Service implementation. Table 6.1 shows how long it took, in a 450 MHz Pentium II, to process 3,000 distributed events (of seven different types with four primitive attributes each) when the knowledge base of the embedded Jess inference

engine was loaded with 76 rules and 128 facts on average. The ECA Service managed to process around 420 events per second in the extreme case tested, where one in three events inserted provoked the firing of a rule. In practice, a much smaller ratio between events inserted and rules fired is usually expected.

Facts inserted	3,000
Rules within engine	76
Rules fired	922
Average number of facts in engine	128
Total time in seconds	8.441
Events processed per second	419.87404

Table 6.1. ECA Service's performance results.

6.5 Building Applications with the ECA Service

Consider the development of a sentient jukebox application that determines whether it is suitable to initiate the playback of music for a user, and if so, decides the kind of music to play, according to the user's contextual conditions, *e.g.* day of the week, his activity or location. In order to build such an application, a number of rules describing the behaviour expected by users from the sentient jukebox upon different set of contextual conditions, would have to be specified. An example rule could be:

> *"If it is Monday, a laboratory member is logged in and if he is working or it is raining outside, then play some cheerful music to raise the user's spirits".*

Without the support of a middleware service providing composite event detection, the reactive application would need to register with the event sources that provide the contextual information of interest, *e.g.* people location events, people login events, keyboard activity events and weather condition events. Then, the application would have to correlate all these primitive events to determine when a composite event fulfilling the complex situation would be matched. The implementation of just the rule mentioned above is already rather complex. If several rules of this kind were needed, the application development would become more complicated. Moreover, if the rules associated with an application are changed later, the hard-coded composite event monitoring process would

have to be modified. This is a result of not separating the application logic (situation monitoring) from the functionality implementation (music playback).

```
within 15000 {/* Ensure events occur in 15 secs time span*/
  query PCMonitor$logged_in(user ?userID, host ?hostID) and
  test(dayofweek = "Monday") and
  Location$presence(user ?userID) before
  /* a presence event must occur before any event on its RHS */
  ((PCMonitor$keyboard_activity(host ?hostID, intensity ?i) and
    test(?i > 0.3)) or
   (query WeatherMonitor$report(raining ?rainIntensity) and
    test(?rainIntensity > 0.2)))
=>
  notifyEvent(Jukebox$play_music(?userID, ?hostID, "ROCK"));
}
```

Figure 6.4. PlayCheerfulMusic rule expressed in the ECA language.

```
(assert (rule (ruleID 0) (ruleRegTime 1005472984621)))
(defrule rule0
  (PCMonitor$logged_in (user ?userID) (host ?hostID)
                       (timestamp ?time0#))
  (test (eq (dayofweek) "Monday"))
  (Location$presence (user ?userID) (timestamp ?time1#))
  (test (> ?time1# 1005472984621))
  (test (> ?time1# (- (curtime) 15000)))
  (or (and (and (PCMonitor$keyboard_activity (host ?hostID)
                            (intensity ?i) (timestamp ?time2#))
                (test (> ?time2# 1005472984621))
                (test (> ?time2# (- (curtime) 15000)))
                (test (> ?time2# ?time1#)))
           (test (> ?i 0.3)))
      (and (WeatherMonitor$report (raining ?rainIntensity)
                                  (timestamp ?time3#))
           (test (> ?rainIntensity 0.2))))
=>
  (bind ?currentTime# (curtime))
  (bind ?factID0# (assert (Jukebox$play_music# 0 ?currentTime#
                                        ?userID ?hostID "ROCK")))
  (notify-event ?factID0#))
```

Figure 6.5. PlayCheerfulMusic rule mapped to CLIPS.

Through the use of the ECA Service, the development of the sentient jukebox application is much easier. First, the developer would define a high-level event type that contained the information necessary to trigger the playback of music in a given machine. For example, the application could define a `play_music` event containing as attributes: the `user` for whom music would have to be played, the `host` where the music should be

played and the `kind` of music to play. Secondly, the application would register the metadata associated with this new type of event with the Event Type Repository belonging, for instance, to the domain `Jukebox`. Finally, the developer would create rules, containing as LHS, a set of conditions over events that should be monitored and as RHS a notification action reporting the `play_music` aggregated event. Figure 6.4 shows the definition of such a rule in the ECA language, whereas Figure 6.5 depicts how this expression is mapped to CLIPS by the ECA language compiler. An analogous process could be carried out to define new rules for other situations upon which music playback should also be started. Likewise, rules triggering a `stop_music` aggregated event when the set of current contextual conditions so indicate should be defined, *e.g.* someone important is approaching our room.

Upon reception of an ECA rule, the ECA Server will delegate the parsing and mapping of the rule to CLIPS to its rule registration module. If parsing of the rule is unsuccessful, the server reports the errors to the client. Otherwise, the rule registration module undertakes a run-time event type checking on the contents of the specification. If it does not know about certain types of events, it will contact the ETR to retrieve the metadata associated with the unknown event types. The ECA Server caches, internally, the retrieved event type descriptions for later use. Next, the rule registration module registers with all the event sources supplying the requested event types in the rule. Finally, this module inserts the ECA rule mapped to CLIPS into the embedded CLIPS inference engine.

The temporal constraints introduced by the ECA language compiler in the ECA-to-CLIPS mapping process of the rule listed in Figure 6.4 are shown highlighted in Figure 6.5. The `within` declaration in the ECA rule results into two CLIPS `test` statements forcing event occurrences not qualified by the `query` keyword to be within a 15 second time span. The `before` keyword in the ECA rule enforces the `presence` event occurrence to temporally precede any event occurrence on its RHS, *e.g.* the `keyboard_activity` occurrence. Finally, the compiler ensures that every new instance of an event pattern not preceded by the `query` keyword has a more recent timestamp than the rule registration timestamp.

Arriving CORBA Structured Events, see Table 6.2, are mapped by the ECA Server's event reception module into CLIPS facts, see Figure 6.6. This module decomposes the contents of arriving events based on the event content descriptions stored in the event type cache kept by the ECA Server. Basically, this process is about flattening the contents of the Structured Event into one or several CLIPS facts.

`Location.presence`		
Member Name	Member Type	Content
`timestamp`	`double`	Milliseconds since 1/1/1970
`user`	`string`	Username of a user, *e.g.* dl231
`room`	`string`	Room ID where user is, *e.g.* "room 1"

Table 6.2. `Location.presence` as a CORBA Structured Event

```
(deftemplate Location$presence
  (slot timestamp)
  (slot user)
  (slot room))
(Location$presence (timestamp 123435) (user "dl231")
                   (room "room 1"))
```

Figure 6.6. `Location.presence` Structured Event mapped to CLIPS.

On successful matching of the situation expressed by the LHS of a rule, the engine will issue requests into either the notification or the action dispatching modules in order to trigger the pertinent actions. In Jess, new functions to the Jess language can be added by simply writing a Java class that implements the `jess.Userfunction` interface. Both the notification and action dispatching modules make use of this interface. They represent gateways into a complex Java subsystem from the internals of the Jess inference engine. Jess extensions used by the compiler's CLIPS code generation, such as `dayofweek` or `notifyEvent`, shown in Figure 6.5, also implement this interface.

6.6 Related Work

The ECA concept is inspired by previous work in the Active Databases [Dittrich+96] [Paton+99] research domain. An active database continually monitors the database state (including the system clock) and reacts spontaneously when predefined events occur. Functionally, an active database management system monitors *conditions* triggered by database *events* (for instance, updates) and if the condition evaluates to true, the *action* is

executed. This work addresses a similar problem but in this occasion the ECA concept is applied to the correlation of the influx of vast volumes of real-time events.

A key factor that distinguishes this work from previous research in Composite Event Management [Bacon+00][Gruber+99][Carzaniga+01] is that the ECA Rule Matching Engine makes use of the pattern matching capabilities of the CLIPS programming language to undertake the complex rule-based reasoning. Other approaches required a complete implementation of the matching engine from scratch. For instance, [Gatziu+94] used Petri Nets, [Bacon+95] finite state machines and [Samani+97] a tree-based mechanism for the implementation of the event composition engine. [Bacon+00] and [Gruber+99] suggest capable event algebras for the specification of sophisticated composite event constraints. Nevertheless, the ECA language expressive power is richer, permitting the specification of action commands in the RHS of a rule without being restricted to just event notification actions.

The Generalised Event Monitoring Language (GEM) [Samani+97], similarly to the ECA language, addresses event composition and aggregation in distributed systems. A distinctive feature of GEM is its built-in capability to deal with the variable delays between the occurrence and the detection times of an event, due to network delay. A GEM script declares events classes, rules defining the actions to be taken when an event is triggered and commands triggering events and disabling or enabling rules. GEM's main drawback is its adoption of a proprietary event format with support for only primitive data types. The rule activation time span concept used by the ECA language was inspired by Samani's *detection window* concept, *i.e.* the sliding time window within which the event monitor and therefore the rules may operate.

Erlang [ERICSSON01] is a virtual machine based functional programming language devised to make programming of real-time concurrent systems easy while remaining efficient. It is argued that this language focuses on facilitating the production of distributed soft real-time systems by helping the programmer to concentrate on the implementation of the functionality, without having to deal with the low level details of distributed communication and concurrency handling. The ECA language shares the simplicity goal of Erlang, but it is a much simpler language concentrated on providing constructions for composite event handling, rather than the broader scope of Erlang.

Other researchers in context-aware computing have also identified the need for an infrastructure to deal with context-based triggering of actions. In particular, the Rome [Huang+99] and stick-e note projects [Brown+97] use the concept of triggers. A trigger is some action that should be taken when a previously stated condition is satisfied. The Rome project's most attractive feature is that it introduces a decentralised evaluation of triggers. Nevertheless, this project limits the constraint specification to temporal and spatial contexts. The stick-e notes project proposes an XML-based language for the specification of the constraints upon which a unique action, *i.e.* presentation of information, should be triggered. This approach enables computer illiterate people to program context-aware applications. However, the applications that can be defined are very simple and their context algebra is rather limited.

The AMIT project [IBM01] defines a sophisticated XML-based language for the specification of event correlation situations with a Situation Manager component. This system can only return the set of events matching a situation; no support for aggregated events notification or other types of actions is provided. The architecture of AMIT's Situation Manager is similar to the ECA Server's one.

The Apama Matching Engine [Bates01] is probably the first commercial product offering real-time composite event monitoring. In Apama, applications set up *monitors* defining queries that should be evaluated against the ever-changing streams of data. When the engine encounters a sequence of data patterns that corresponds to a previously defined monitor, the engine sends an *alert* to the application. The concept of monitors is equivalent to the ECA rules. Apama's declarative language permits the generation of concise event correlation specifications describing time-related, location or content-based matches. However, constraints in the Apama language consist of only the situation-part of an ECA rule. The default action is fixed to be an event batch notification containing the event instances matched. Apama's event correlation engine is implemented by dynamic state machines rather than leveraging from a rule-based language's inference engine as in the ECA Service. The Apama engine is focused on addressing scalability, matching thousands of monitors against huge quantities of event data. The ECA Service provides a simpler-to-grasp language and model with which to develop reactive applications, although it is not as scalable as the Apama engine.

6.7 Conclusion

The ECA Service makes the development of reactive (sentient) applications a less cumbersome process by performing the event correlation process on behalf of applications. The ECA language enables the creation of sophisticated rules, expressing complex conditions upon the primitive events that are part of a high level composite event, and the actions to be taken when those conditions are met. On the action side of a rule, a programmer may specify: (1) a high-level aggregated event or (2) the batch of events corresponding to the matched situation to be notified, or, alternatively, (3) a set of pre-defined or user specified actions to be fired. This approach relieves the programmer from the tedious and difficult implementation of composite event monitoring. Programmers simply concentrate on the specification of (contextual) situations and delegate their matching to the ECA Service; only having to implement the reactive behaviour desired when notifications of the matched situations are received. The ECA Service represents this work's solution to the very difficult task of making computers 'understand' the current contextual situation surrounding them.

Chapter 7

Object Lifecycle and Location Control

Computing devices are increasingly becoming ubiquitous. Computationally capable devices are present in almost every room of a modern office, and are rapidly invading our homes. Moreover, those environments are becoming aware; sensor systems such as TRIP allow applications to respond to the location (and other types of context) of users. Therefore, the creation of the first *sentient spaces* is becoming viable.

Software support for Sentient Computing (see sections 2.2 and 5.6, and chapter 6) has focused on providing mechanisms to facilitate context capture and abstraction, so that context can be directly used by applications. However, most of the proposed solutions stop their support at the point at which a sentient system must raise an *action* as a consequence of a matched *situation*. This corresponds with the third aspect in the development of a sentient system, *i.e. action triggering*.

In order to promote a wider usage of the Sentient Computing paradigm, it is insufficient to make computing systems *perceive* and *interpret* context in some meaningful form. It is also necessary that these systems have the capability to automatically activate services on behalf of the user when the appropriate contextual conditions are met, *i.e.* to provide some assistance for the action triggering stage of sentient systems. Moreover, it is desirable to enable user-bound services to follow the user as they move through the physical space, and to disable such services when the user leaves the space. As a consequence, sentient spaces can be created where users move freely throughout a building or greater environment without substantial degradation of the computing and communications resources available to them.

Several previous research efforts have addressed the creation of software infrastructures to enable the transparent on demand activation and migration of services associated to a user. However, these are either constrained to enable migration of components written in a specific portable programming language such as Java [Baumann+98][Glass99] or Python [Hylton+97], or present a partial solution, enabling only some of the functionality of the components, such as the GUI [Richardson+98] or the multimedia streams [Bacon+97], to move. This chapter describes a more general CORBA-based solution to the management of distributed objects' lifecycles and locations within a network.

7.1 Component Mobility in CORBA

The CORBA [OMG01] object model relies heavily on the semantics of object references. The standard object reference format, namely the Interoperable Object Reference (IOR), contains an identifier of the most derived type of an object and one or more *profiles* that permit the location of an object to be determined. Among other data, these profiles may contain an IP address, a TCP port number, and an *object key* that uniquely identifies the target object. Clients can only invoke an operation on an object if they hold a reference to it. Unfortunately, the location-specific information in an IOR, *i.e.* the IP address and port number of the server at which the object resides, implies that when a CORBA server implementation moves, the clients can no longer maintain the connection to the server.

Adding mobility support to CORBA supposes that any client holding a reference to an object will maintain that binding for the entire lifetime of the object. This means that client invocations must be forwarded seamlessly when the location of the targeted object changes. CORBA provides the means to handle this situation by defining within GIOP [OMG01] the `LOCATION_FORWARD` reply message. A client ORB receiving this reply message must retry a previously issued request using the enclosed IOR.

7.2 The LocALE Middleware

The LocALE[18] (Location-Aware Lifecycle Environment) framework defines a simple mechanism for managing the lifecycle and location of distributed CORBA objects residing on a network. In addition, it provides load-balancing and fault-tolerance features to the objects whose lifecycles it manages. The emphasis of its design is placed on providing a suitable interface for third party *object-location controller*, *e.g.* frameworks for software adaptation [Edmonds01] or sentient applications. These controllers can intelligently direct when and where components are activated or moved based, for example, on the inputs on user whereabouts and the location, load and capabilities of computing resources. LocALE aims to offer an object-lifecycle handling infrastructure useful for the action-triggering aspect of sentient systems. Moreover, LocALE simplifies sentient application deployment and enables CPU intensive systems, such as TRIP, to efficiently use the spare computing resources in a network.

LocALE addresses the following functional requirements:

- *Permit the remote instantiation of CORBA services wherever they are needed*. Traditionally, in a distributed environment, the factory design pattern is used to avoid having to start servers manually every time a client requires a service. A factory is an object that offers one or more operations to create other objects. Upon client invocation, the factory creates a new object and returns a reference to the client. Conventional factories normally constrain clients to creating new objects within the address space of previously started factories. It would be desirable for a client to be able to control the location (*e.g.* host) where a new service is instantiated, independently of whether there is already a factory capable of creating objects of the desired type at that location or not.

- *Enable heterogeneous object migration*. Computational distributed objects (part of an application) may have to migrate to optimise speed and latency, or to get physically closer to a user. Communication endpoints may consequently change location over time. An infrastructure providing transparent object migration to clients without

[18] The author is indebted to Jamie Walch for suggesting the name LocALE.

affecting the references they hold is required. Moreover, objects written in different programming languages and running in different platforms should be able to migrate.

7.2.1 LocALE Migration Support Rationale

This section discusses the design decisions adopted by LocALE in order to provide migration support for CORBA components. Migration of software objects can be understood as a two-phase process: (1) transfer state and, optionally, implementation of a running object to a new location and (2) cause all clients of the object to transparently use the new object.

With regard to the first aspect, a migration system must decide whether the system should support both code and state migration, or simply state migration. The use of portable, virtual machine-based, programming languages, such as Java or Python with built-in object serialisation and bytecode portability features, facilitates object state and code migration. However, LocALE aims to control the lifecycle of heterogeneous objects, without being constrained to a single programming language or platform. Thus, LocALE does not support object implementation transfer upon object migration. It is desirable to migrate objects to different implementations of the same functionality. For example, an object implemented in C++ running on a Windows NT host should be able to migrate to a Java version of it running in Linux. This signifies that wherever an object moves there must be access somehow, *e.g.* locally or through a network file system, to a valid object implementation.

There are two basic methods by which state transfer can be performed. One option is to take a snapshot of a running object and transfer that state. Java's object serialisation [SUN98] permits this to some extent. The main problem of this approach is that it prevents objects from moving to different implementations, *e.g.* a C++ object cannot be migrated to a Java object version of it. A second option is for objects to transfer their own state to a clone of them when they are instructed to migrate. One or several methods of a newly created object are used to set its state, according to the state of the source object. LocALE adopts this approach as it suits heterogeneous object migration and allows a more general interpretation of what can be considered to be 'state'.

The second issue to be addressed when designing an object migration system is how clients of an object can track the object as it moves. CORBA provides support for this by the GIOP's `LOCATION_FORWARD` message. There are two obvious ways in which this mechanism can be used to track object locations: (1) chaining of forwarding agents and (2) use of a home location forwarding agent.

With the first approach, every time an object migrates, it leaves behind an agent, which forwards clients to the object's new location. The result is a trail of forwarding agents at every location the object has ever occupied. This causes intolerance to faults, and difficulties with garbage collection. If one of the agents dies, the agents forwarded by the disappeared agent would not be reachable anymore.

The second approach nominates, for each object, a *home location* that is guaranteed to know the current location of the object. The home location either provides the object itself, or forwards the client to it. Clients remember the home location, and consult it whenever the object ceases to exist in the location they were using. Whenever an object migrates, it informs the home location of its new location. This is the object tracking mechanism employed with LocALE since it introduces less latency and better resource cleanup properties than the chained approach, although it still presents a single point of failure problem. A solution for this latter problem is offered in section 7.8.

7.2.2 LocALE Architecture

The LocALE middleware has a 3½-tier architecture (see Figure 7.1). It consists of client applications and the following 3 types of components:

- The *Lifecycle Manager* (LCManager). This component provides object-type independent lifecycle control interfaces. It mediates the lifecycle operations over all CORBA objects residing in a location domain. A *location domain* is a group of machines in a LAN that are physically located in the same place, such as a building, a floor or a room. Every object creation, movement or deletion request is routed through this component. This permits the LCManager to cache the current location of every object in a domain and thus act as a *forwarding agent* that redirects client requests after object references held by clients are broken due to either object

movement or failure. In the latter case, the LCManager tries to recover the failed object, and, if successful, returns the new object reference.

- *Lifecycle Server(s)* (LCServer). These components are containers of LocALE-enabled objects that are subjected to lifecycle control. Lifecycle operations invoked on an LCManager are delegated to suitable LCServers. They subsume standard strongly typed factory object functionalities by providing a type specific creation method with hard-coded types. Furthermore, they also assist local objects on their migration to other LCServers. They can be started either manually or by the local LCManager after a creation or migration request arrives at a location where the required LCServer type does not exist. In either case, they register with the manager by passing on their physical location (host), their CORBA object reference and data specific to the type of objects they handle (*e.g.* name and parameter names and types of the method which enables the creation of the type of objects they host).

- *Type-specific Proxy Factories.* These components are placed in between clients and LCManagers. They prevent client applications from dealing with the generic object creation interface offered by an LCManager. Using type-specific interfaces makes client code shorter and simpler to understand. With the generic version, if a mismatch in the type of a constructor argument occurs, the client would only receive an error at run-time rather than at compile time. *Type-specific Proxy Factories* offer type specific object-creation methods with the same argument types and semantics as the LCServers they represent. Their mission is to map type specific object creation requests to the generic format demanded by LCManagers. They are activated on-demand by the local LCManager.

The LocALE architecture guarantees compile-time type safety of clients with respect to the lifecycle control of the services they use. Clients use the LCManager to find object references to type-specific proxy factories and issue object creation requests through them. Object migration and deletion requests are directly invoked on the LCManager because they are object type independent. The LCManager then delegates incoming lifecycle operations to the appropriate LCServers. As depicted in Figure 7.1, the LocALE architecture consists of four logical layers. However, only three real physical layers exist

in its implementation, since Proxy Factories are actually contained within LCManagers. This is the reason for stressing the 3½-tier nature of the LocALE architecture.

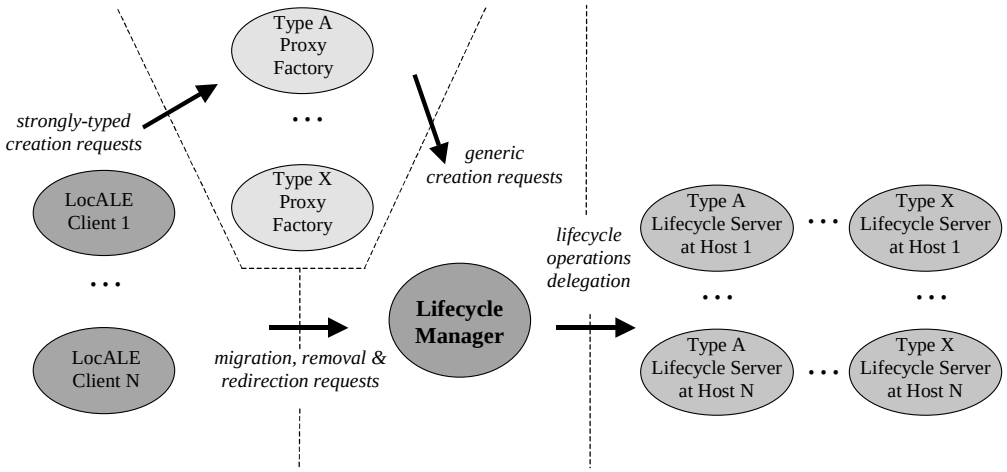


Figure 7.1. LocALE 3½-tier architecture.

7.2.3 Location-constrained Lifecycle Control

One of the main advantages of CORBA is that it offers location transparency to applications, *i.e.* it makes method invocations as simple on remote objects as on local objects. In contrast, LocALE provides applications the capability to control *where* their associated services are located and can also set the constraints under which these services can later be re-located. This work's view is that these additional functions bring important benefits for certain types of applications. For example, *load-balancing* systems may wish to initiate new service instances on the hosts of the same LAN with the lowest processing load. Follow-Me sentient applications may want to move objects tied to a user's physical location to the nearest host with the required capabilities. In essence, LocALE provides client applications with *location-awareness* of the CORBA services they use. In order to allow applications to control the lifecycle and location of objects, LocALE extends conventional object factories with additional distributed object construction semantics. This means that:

- Clients have to specify not only the arguments of the constructor of an object, but also the *location* where the object should be created and some *constraints* controlling its life evolution.

- Object creation requests are always routed through the local domain's LCManager that finds or creates suitable LCServers where such requests are delegated. Thus, the manager can cache the object references returned by surrogate LCServers and convey to clients specially manufactured IORs tied to objects for their entire lifetime.

The following two attributes are appended to both the type specific object creation interface provided by proxy factories and the generic interface offered by LCManagers:

- The *location* attribute specifies *where* a service should be instantiated or moved to. Its possible formats are shown in Table 7.1.

- The *lifeCycle contraint* attributes determine whether an object that is being created should be considered as recoverable and/or movable within a location domain. A *recoverable* object is either a stateless or a persistent object whose state can be restored after failure. A *movable* object knows how to transfer its own state to a clone of itself. Table 7.2 lists all existing lifecycle constraints and their usage dependencies with location attributes. For example, the `RECOVERABLE_WITHIN_ROOM` constraint can be applied if and only if a location attribute corresponding to a specific room or a specific host was used.

`hostDN("Guinness.eng.cam.ac.uk")`	Instantiate new object in host guinness
`hostDN("ANY")`	Instantiate new object in any host of the domain
`roomID("Room 1")`	Instantiate new object in any host within Room 1
`roomID("ANY")`	Instantiate new object in any host of any room
`hostGroup(hostDN_1, …, hostDN_n)`	Instantiate new object in one of the given hosts

Table 7.1. Location attribute specs.

```
RECOVERABLE | ∀ Location specs
RECOVERABLE_WITHIN_ROOM | iff (Location = roomID(x) ∨ Location = hostDN(x)) ∧
                          ¬(x = "ANY")
RECOVERABLE_WITHIN_HOST | iff (Location = hostDN(x)) ∧ ¬(x="ANY")
MOVABLE | ∀ Location specs
MOVABLE_WITHIN_ROOM | iff (Location = roomID(X) ∨ Location = hostDN(x)) ∧
                      ¬(x = "ANY")
```

Table 7.2. Lifecycle constraints and usage rules.

Location independent services are instantiated using `hostDN("ANY")` or `roomID("ANY")` constructions. These specifications are of special interest when there

are Lifecycle Servers of the same type running on separate hosts of a location domain. Upon object creation or movement, as a crude form of load balancing, the LCManager randomises the list of available LCServers to select one to delegate the request to. Certain services, however, need to be created in the same location as a user. For instance, an object with a user interface should be created either in the host closest to a mobile user, *e.g.* `hostDN("guinness")`, or otherwise in any host within the same physical container, *e.g.* `roomID("Room1")`. Finally, some services should be created in any machine within a given location that provides a special resource (*e.g.* sound or video capture capabilities). The construction `hostGroup` is used in this case. It is up to the client application or other services to embody the intelligence that selects a suitable list of hosts.

LocALE assumes that objects are, by default, static and non-recoverable. Once an object is created, it remains in the same address space until it is removed or fails. However, LocALE provides mechanisms to make objects `MOVABLE`, providing they know how to transfer their own state, and/or `RECOVERABLE`, providing they can restore their state after failure. The LCManager recreates failed objects by passing the same constructor argument values as when the object was first constructed, and a flag indicating that the object must be recreated. An object can be simultaneously movable and recoverable, as long as both constraints apply to the same location scope, *e.g.* `MOVABLE & RECOVERABLE` is valid but not `MOVABLE_WITHIN_ROOM & RECOVERABLE`.

The capability of receiving location constraints on object creation means that the LCManager is a minimal Trading [OMG00a] server. The LCManager matches client object creation specifications against the advertised object LCServer types and locations. On a service instantiation, its type, initial location and life evolution constraints are conveyed to the local LCManager. This component searches among the registered Lifecycle Server types for a suitable one where the requested lifecycle operation can be delegated. The trading capabilities of the LCManager could be extended, for example, by forcing LCServers to specify the computing resources demanded by their objects (*e.g.* need for sound capability). Thus, if a client issued a lifecycle operation over a given object type, the LCManager would match both location constraints and object type specific resource demands.

LCServer (factID, *kindID*, *hostDN*, factRef, objCreatedCounter, maxNumObjects, manuallyInit)
Object (objID, *kindID*, *factID*, objRef, locationSpec, paramValues, constraintsLC)
ProxyFactory (proxyFactID, proxyFactRef, manuallyInit)
HostLCServers (hostDN, set<*factID*>, *roomID*, OS)
RoomHosts (roomID, set<*hostDN*>)
KindLCServers (kindID, set<*factID*>, objCreationMethod, paramTypes, start-up-process-desc)
KindProxyFactories (kindID, set<*proxyFactID*>, start-up-process-desc)

Table 7.3. Lifecycle Manager internal state relational schemas[19].

7.3 LocALE Lifecycle Manager

The LCManager is the core component of the LocALE architecture. This centralised server keeps, in an internal registry (see Table 7.3), information about the objects whose lifecycle it controls, the proxy factories used for strongly-typed object creation and the LCServers where object creation (activation), migration and removal (deactivation) actually takes place. This information, serialised to disk to permit the LCManager component to recover from transient failures, is used to provide the following functionality:

1. Load-balanced object lifecycle control.
2. Object reference forwarding to clients on method invocation to objects whose location has changed.
3. Automatic activation of LCServers when demanded at a location where they are not available. The LCManager automatically activates LCServers in UNIX platforms[20] by using the `ssh` command. For instance, `ssh heineken EchoLCServer`, activates an LCServer of type `Echo` in host `heineken`.
4. Deactivation of automatically started LCServers.
5. Activation of type-specific proxy factories upon arrival of an object creation request.

[19] Note that underlined attributes denote primary keys whereas italicised members are foreign keys, in a relational databases notation.

[20] Automatic activation in Windows platforms could be possible providing an activation daemon is run in every machine where service activation could occur.

6. Fault tolerance of stateless or persistent LocALE objects by recreating them upon a failed client request.

7.3.1 Swizzling Mechanism for Object Reference Management

The LCManager *swizzles* object references returned by LCServers' creation methods into object references pointing to it. The actual object reference is stored in the `Object` relation (see Table 7.3.). The swizzled IORs, returned to clients, are tied to their corresponding objects for their entire lifetime no matter how many times the objects migrate or are recovered. As a result of this procedure, the first request invocation on an object reference is not addressed to the actual object but to the LCManager. The manager then forwards the client the actual object reference through a `LOCATION_FORWARD` reply. The client ORB transparently retries the operation on the new object reference, and will use this reference while it remains valid. When an object moves or fails, a client request will fail and its ORB will fall back to the original *swizzled* reference. In that case, the LCManager either forwards to a new reference or tries to recover a failed object and return its reference, or raises an error exception. The cost of this mechanism is that client requests are delayed due to the additional round-trips introduced after a lifecycle or failure event occurs. The mechanism is transparent to applications. The payoff is that the LCManager presents object's fault-tolerance and mobility support facilities without breaking client held references. Figure 7.2 depicts the client request redirection mechanism implemented by the LCManager.

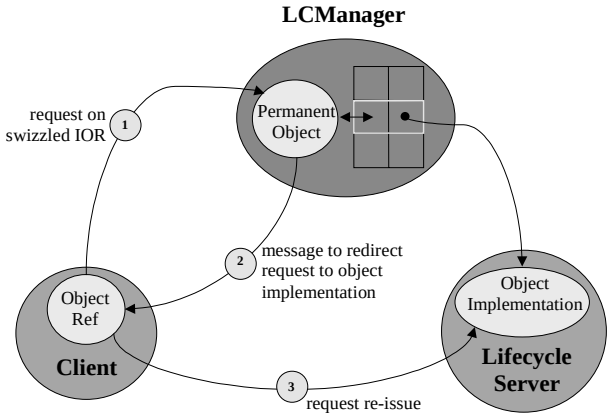


Figure 7.2. Client request redirection.

7.3.2 LCManager Implementation Details

Internally, the C++ implementation (using the omniORB [Lo+00] CORBA ORB) of the LCManager utilises three POA [Schmidt+97] components, namely `LCManagerPOA`, `RedirectionPOA` and `ProxyFactPOA`. Figure 7.3 illustrates the internal architecture of the LCManager.

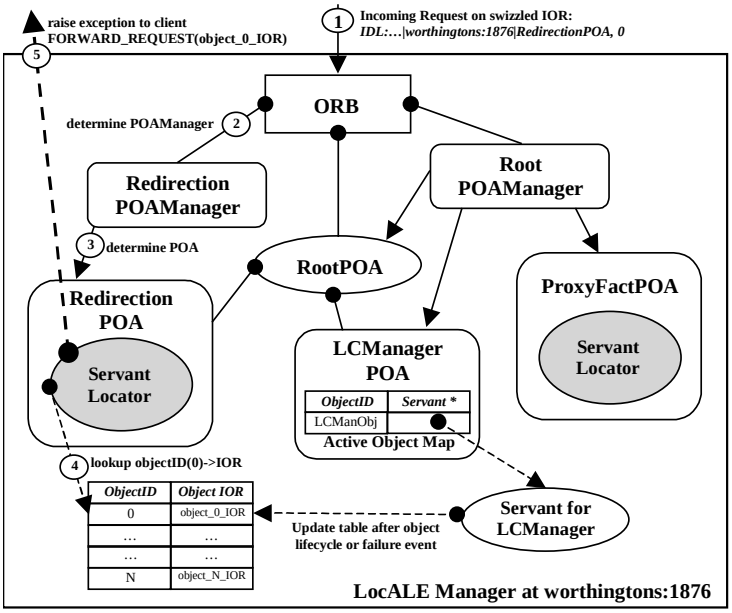


Figure 7.3. Request processing flow over swizzled IOR.

The `LCManagerPOA` contains the servant implementing the remote interfaces offered by the LCManager. The `RedirectionPOA`, which has a Servant Manager [Schmidt+98] object registered with it, implements the client request redirection procedure described above. Client requests on swizzled IORs (created using the `RedirectionPOA`) arrive at this POA, which up-calls a method of its associated Servant Manager implementing client request redirection. This method uses a standard mechanism to generate `LOCATION_FORWARD` reply messages, namely the `ForwardRequest` exception. Finally, the `ProxyFactPOA` is used to create Proxy Factory objects on the fly as DSI servants which map client issued compile time safe creation requests into the generic format required by the LCManager servant. All proxy factories in a location domain are physically activated within an LCManager's address space.

7.4 LocALE Lifecycle Servers

Every Lifecycle Server implements the `LCServer` IDL interface shown in Figure 7.4 and, in addition, defines a method to create objects of a specific type. The LCManager uses the `move_object` method in this interface to trigger the migration of an object under the control of an LCServer to a clone of that object in another LCServer. Every object managed by an LCServer is derived, in turn, from the `LocALE_Object` (also in Figure 7.4) IDL interface. Upon the reception of a `transfer_state` call, an object conveys its state to the clone of it passed as a parameter. Object type specific method(s) are invoked to adapt the state of the target object to the source one.

```
interface LCServer {
 void move_object(in LocALE_Object objRef,
                  in LocALE_Object cloneObjRef)
            raises(UnKnownObj, WrongObjType, InvalidClone);
 void destroy(); // Method to destroy LCServer
};
interface LocALE_Object {
 void destroy();
 void transfer_state(in LocALE_Object cloneRef)
             raises (InvalidClone);
 readonly attribute HostDN currentLocation;
};
```

Figure 7.4. LocALE LCServer and object IDL interfaces.

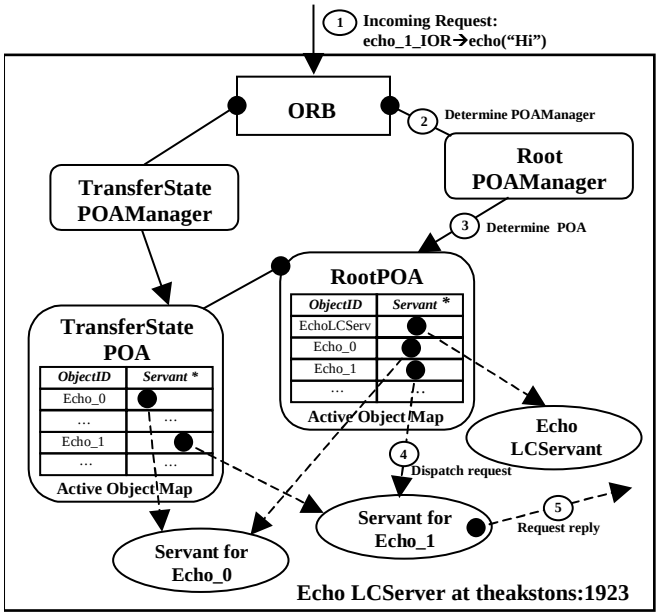


Figure 7.5. Request dispatching in an LCServer.

To support mobility, objects created within an LCServer must be activated simultaneously with two POAs (see Figure 7.5). The `RootPOA` (or any other POA) through which requests from external clients arrive to the LCServer's servant and to servants of the objects whose lifecycle it controls and a second POA (`TransferStatePOA`) through which `state_transfer` operations on the controlled objects are issued by the LCServer. The reason being that on object migration, while the LCServer triggers the state transfer between the source and target servant and waits for its completion, it must guarantee that requests on the migrating object are queued for later processing. Every POA contains an associated POA Manager [Schmidt+98] object that controls the flow of requests into it. Thus, client requests on the migrating object are blocked by calling the `hold_requests` operation on the `RootPOAManager`. The LCServer uses a second IOR for the object, obtained from its `TransferStatePOA`, to invoke the `state_transfer` operation. Once the object migration is finished, the `RootPOAManager` is reactivated. Queued invocations on that POA Manager then result in `OBJECT_NOT_EXIST` exceptions. Client ORBs react to these exceptions by re-routing the requests, assisted by LCManager's redirection capabilities, to the new object locations.

```
interface LCManager {
  LocALE_Object create_object(in Kind k, in AnyList params,
           in Location where, in ObjectLCPolicyList policyList)
      raises (NoLCServer, InvalidParam, NoExistingLocation,
              WrongPolicySpecification, ObjectLimitReached);
  void move_object(in LocALE_Object objRef, in Location where)
      raises (UnKnownObject, NoExistingLocation,
              NoPossibleMigration);
  void remove_object(in LocALE_Object objRef)
      raises (UnKnownObject);
  LocALE_ProxyFact find_proxy_factory(in Kind k)
      raises (UnknownType);
};
```

Figure 7.6. LCManager's object licecycle control IDL interfaces.

7.5 LocALE Object Lifecycle Management

This section shows the interactions occurring among the members of LocALE's architecture when a client issues lifecycle control operations on an object. It also shows

the C++ code required for clients using the LCManager's API (see IDL interfaces in Figure 7.6) to trigger such operations.

7.5.1 Remote Object Construction

The C++ code below illustrates how a client can instantiate a new remote service of type `Echo::EchoObject` at host `heineken`, and invoke an `echo` operation on it.

Figure 7.7 depicts the interactions involved within the LocALE architecture. Observe that in iteration 7 the LCManager has to use DII to create an `Echo` object. This is because the LCManager does not have compile time knowledge of the objects whose creation it mediates.

```
// Create a new recoverable and movable echo object in host
// heineken and invoke echo()
LocALE_ProxyFact_var pFact = LCManager->find_proxy_factory(
                                        "IDL:Echo/EchoObject:1.0");
Echo::ProxyFact_var echoPFact = Echo::ProxyFact::_narrow(pFact);
Echo::EchoObj_var echo1Ref = echoPFact->create_echo_object(
                             "Echo_1",
                             LOCATION(HOST("heineken")),
                             LC_CONSTRAINT(RECOVERABLE¦MOVABLE));
echo1Ref->echo("Hi");
```

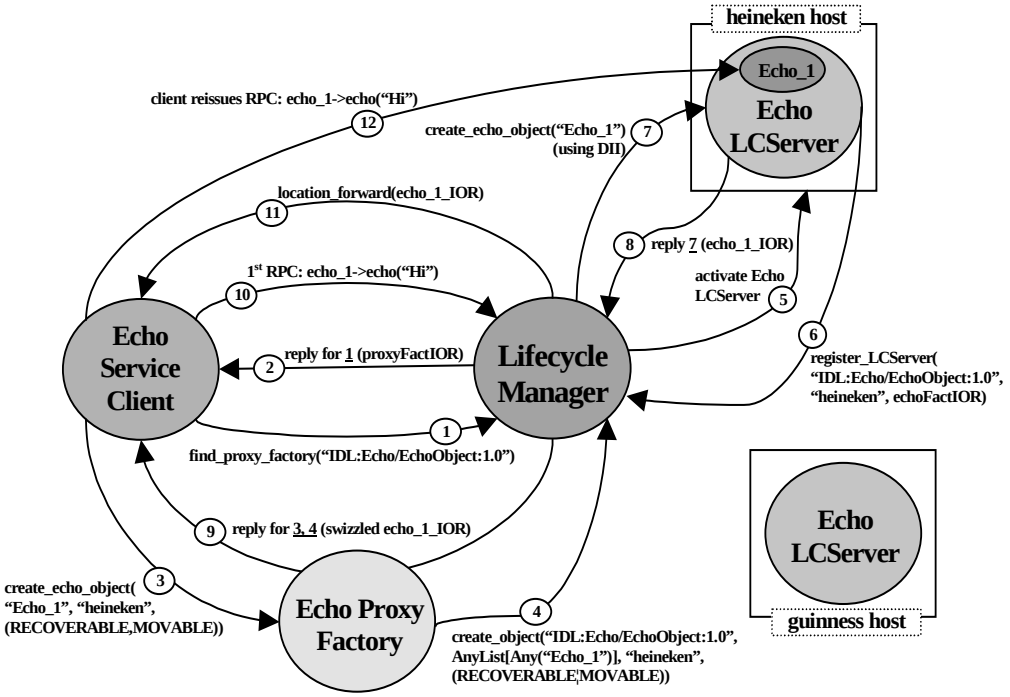


Figure 7.7. Remote object construction in LocALE.

7.5.2 Object Migration

The client code necessary to migrate the previously created `Echo` object to host `guinness` is shown below. The set of interactions involved within LocALE are shown in Figure 7.8. Note that the RPC request in iteration 10 will actually be routed first to the `EchoLCServer` at host `heineken` and after its failure, as the `Echo` object will not reside there anymore, it will be rerouted by the client ORB to the LCManager.

```
// Move object to host guinness and invoke echo() on it
LCManager->move_object(echo1Ref, LOCATION(HOST("guinness")));
echo1Ref->echo("Hi again");
```

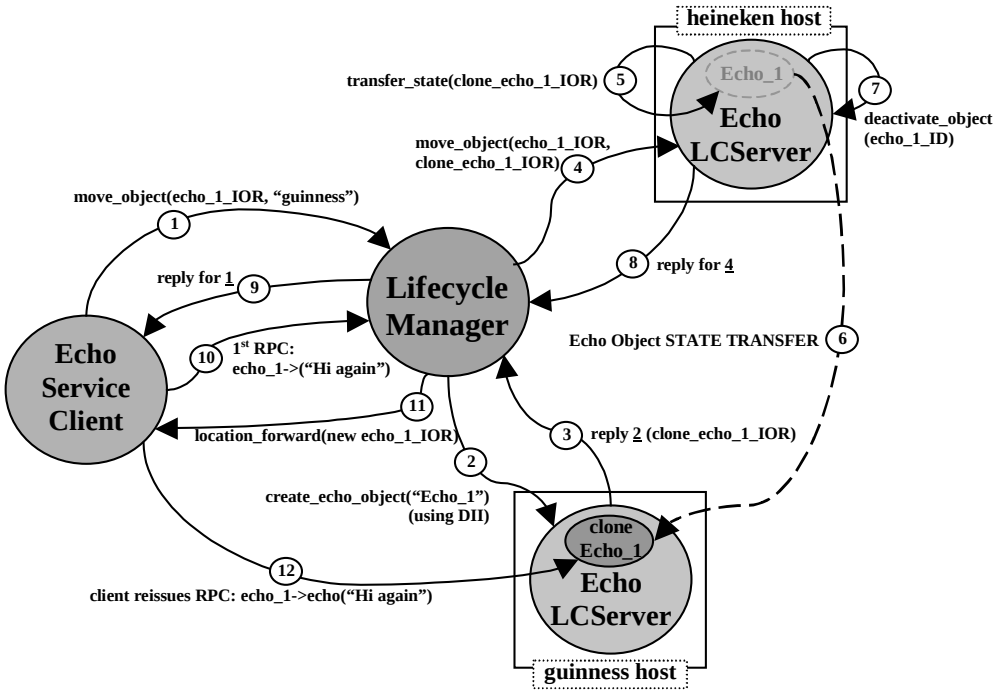


Figure 7.8. Object migration in LocALE.

7.5.3 Object Deactivation

The client code necessary to deactivate the previously migrated `Echo` object from host `guinness` is shown below. Note that iteration 5 will actually be directed first to the `EchoLCServer` in host `guinness` and fail. The set of interactions involved within LocALE are shown in Figure 7.9.

```
// Remove the echo object moved to guinness
LCManager->remove_object(echo1Ref);
// The following call raises an OBJECT_NOT_EXIST exception
echo1Ref->echo("Hi after removal");
```

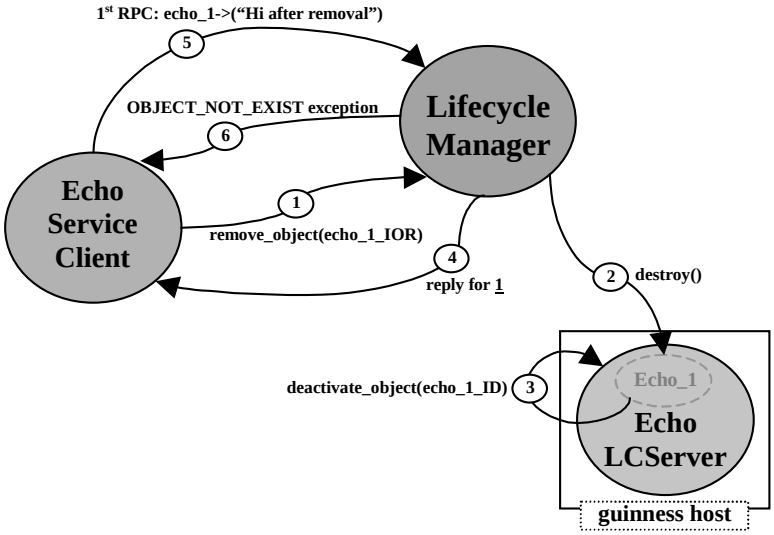


Figure 7.9. Object deactivation in LocALE.

7.6 Applications

This section describes two LocALE-enabled applications built to demonstrate LocALE's potential as an enabling infrastructure for Follow-Me applications. Both applications are composed of multiple distributed objects. When a user moves, only those objects related to the user's physical location follow him. All inter-object communication is through CORBA RPCs.

7.6.1 Follow-Me Audio

The Follow-Me Audio application supplies mobile users with music from the nearest set of speakers, wherever they move in the LCE laboratory. The source of music is a static MP3 jukebox server that is controlled using the buttons of an Active Badge or special TRIPtags. A personal software agent, associated with each member of the laboratory, listens for that person's movement events coming from the Location Service Context Abstractor described in section 5.7. This agent uses LocALE migration capabilities to

move the audio player and MP3 decoder components of the application to the host supplying the appropriate facilities that is nearest to the user. As the user moves around the location-aware environment, the music is played back continuously from the nearest set of speakers. The state of the system and time index into the current song persists as the components migrate. Section 8.3.1 provides a more thorough description of this application, showing how TRIP, the ECA Service and LocALE were used for its implementation.

7.6.2 Follow-Me Email Notification

Another personal agent has been built which acts as an IMAP mail client and an event sink for location events coming from the Location Service Context Abstractor. This agent remembers the last location where a user was seen. When an email for that person is received, the agent remotely instantiates, through LocALE, a text-to-speech object at the host (with sound capabilities) closest to the user. A verbal notification with the sender and subject details of the message is then produced from the nearest speaker. The body of the email message may also be read on the user's request, by clicking one of the Active Badge buttons or showing a special TRIPtag. After a timeout expires, the agent deactivates the text-to-speech object through LocALE.

7.7 Related Work

The CORBA Object Life Cycle Service [OMG00] defines conventions that allow clients to perform lifecycle operations on distributed objects. It provides IDL interfaces that permit clients to control the lifecycle of objects without knowing about their types or semantics. This makes this service suitable only for applications that require a weak type model. LocALE subsumes the functionality of this service, still guaranteeing client code compile-time type safety through its proposed 3½-tier architecture.

The Jumping Beans [Aramira01] software infrastructure provides developers the ability to create CORBA applications that physically jump from one device to another while the application is running. This framework is based on a centralised Jumping Beans server. When a CORBA server object moves, it first moves to the Jumping Beans server and then to its final destination. Each host to which mobile applications can be migrated must run a

Jumping Beans client daemon that listens on a known TCP/IP port to receive mobile applications, and to guarantee security. This project resembles LocALE in the use of a central entity that controls object migration. In contrast to LocALE, it does not address heterogeneous component migration. It only provides mobility support for Java implemented objects but can hence support implementation code transfer and address the important security issues regarding service migration. LocALE also differs from Jumping Beans in its capability to perform location constrained object lifecycle control. Finally, LocALE provides fault recovery of CORBA servers on client request arrival, whereas Jumping Beans concentrates on providing fault tolerant component migration by using a store and forward with guaranteed delivery technique.

InConcert [Brumitt+00][TIBCO00] is a middleware solution that addresses the following issues: (1) asynchronous message passing, (2) machine independent addressing and (3) XML-based message protocols. LocALE does not address asynchronous message passing since that is the task of orthogonal services such as the CORBA Notification Service. Machine independent addressing is achieved in LocALE by means of the creation of persistent object references that are valid for the entire lifetime of the object, no matter how many times it changes location. InConcert argues that the main advantage of using XML for message encoding is that it permits the evolution of the interfaces of servers without affecting the code of clients complying with the original interface. A similar behaviour can be achieved in CORBA if the new server interface is derived from the original one. InConcert does not provide support for location-constrained on-demand activation, migration or destruction of services, as in LocALE.

The DAWS project [Grisby99] proposes a decentralised heterogeneous component migration framework where, when an object migrates from the location where it was created (*home location*) to another location, it leaves behind a *forwarding agent*, which emits `LOCATION_FORWARD` messages pointing to the new location. This design presents good scalability properties. In both LocALE and DAWS, objects are responsible for transferring their own state during object migration. Furthermore, these frameworks do not support code migration and require at each host compiled-in support for all objects they can serve. DAWS is ORB-dependent because its implementation modifies a CORBA 2.0 ORB, lacking POA support. LocALE's work has been built on top of a CORBA 2.3 ORB and, therefore, can control the lifecycle of objects using any vendor's

ORB by means of standard POA features. Finally, DAWS does not support the load-balanced remote service instantiation, LCServer automatic activation and deactivation, and object recoverability features available with LocALE's centralised design.

A LocALE LCManager is *per se* a CORBA Implementation Repository (IMR) [Henning98]. Similarly to an IMR, an LCManager provides functionality to (1) track CORBA objects' current location, (2) forward a client request for an object to the correct process, and (3) automatically activate servers on demand. But in addition, LCManagers also control object lifecycle operations. Existing IMRs only cache location and activation information of server object adapters (POAs) where objects are activated. This offers good scalability because the same adapter information is used to provide indirect binding support to all objects contained within it. Nevertheless, it only enables migration, at once, of all objects registered with a particular object adapter. Furthermore, IMRs are ORB-specific because they can only manufacture forwarding IORs for objects created with their same ORB. Both IMRs and LCManagers share the common problem of constituting a single point of failure and bottleneck. Moreover, support for object migration into a different location domain has not been tackled in either LocALE or in IMRs.

Mobile Agent systems, such as Voyager [Glass99] or Mole [Baumann+98], are focused on enabling the transparent migration of agents, including their code, data and threads, from place to place. Whilst LocALE clients direct objects' lifecycle, mobile agents are autonomous and detached from clients. LocALE is designed to be used within a LAN where security is less of an issue; mobile agents are typically used on the Internet where security and reliability are much more important. Mobile agents embody the intelligence necessary to decide when to move to another host, whereas in LocALE that intelligence must be provided by client applications.

7.8 LocALE Shortcomings

There exist two main limitations associated with the design of LocALE: (1) the Lifecycle Manager constitutes a single point of failure and bottleneck within a location domain and (2) an undesirable overhead is incurred on the first RPC over a LocALE object.

The first problem may be addressed by replicating the LCManager and using, for instance, group communication mechanisms [Maffeis96] to synchronise the internal state of the LCManager replicas. Multi-profile IORs embedding all object references of the federated LCManagers could then be manufactured upon object creation. Thus, clients' ORBs would transparently re-issue a request on the object reference of another replicated server when one member of the federation failed [OMG01c]. Likewise, the load balancing features of the system could also be improved by selecting, in a random fashion, one of the multiple LCManager IORs available.

Scalability in LocALE can be compromised since an LCManager handles every object lifecycle/redirection operation within a location domain. However, once an object has been activated, migrated, deactivated or recovered, and then, forwarded for each client, no further access to the manager is required. From then on, the client directly accesses the object without further mediation by the manager. The overhead incurred on the first RPC over an object, after a lifecycle or failure event occurs, may be solved by including the actual object reference as the primary profile within a multi-profile IOR [OMG01b] returned to the client. The overhead in the first RPC is especially significant if it involves large amounts of data. To solve this, the object can be pinged before the actual request is issued; the `non_existent` operation available in every CORBA object can be used.

7.9 Conclusion

This chapter has described the design of LocALE, a framework for the management of CORBA objects' lifecycles and locations in a fixed ubiquitous network. LocALE permits client applications to constrain the physical location of newly created objects, and to define the location restrictions under which objects may later be migrated or recovered. Object lifecycle requests issued by clients are routed through a central Lifecycle Manager that, after matching objects location and lifecycle constraints, delegates requests to suitable LCServers. Likewise, the LCManager redirects client requests after an object's location changes. The capability of LocALE as an enabling infrastructure for location-aware mobile computing has been demonstrated by the successful implementation of several Follow-Me applications. A suitable application area for LocALE, not tackled in this dissertation, is the development of load-balanced distributed systems. LocALE

addresses the need encountered in section 5.9 for a software infrastructure to help developers in the implementation of the action-triggering aspect of sentient systems. LocALE simplifies the deployment of sentient systems across a networked environment, and enables the use of spare computing resources available at that network.

Chapter 8

Applications

The previous chapters have described three main contributions that are deemed to be essential to enable a more widespread adoption of the Sentient Computing paradigm in our living and working spaces. First, a low-cost and easily deployable vision-based location technology, termed TRIP, was introduced. Secondly, the ECA Service, a middleware to which sentient systems can delegate pre-defined contextual situations monitoring, was outlined. Thirdly, a middleware aiding the action-triggering aspect of sentient systems, named LocALE, was described. This chapter focuses on demonstrating how this assortment of technologies permits the development and deployment of useful sentient applications. Moreover, they do so in a very easy and effective manner for developers, incurring a very reduced cost and almost no additional hardware investment. The applications presented are categorised by the way in which they use the TRIP technology.

8.1 Off-line Processing of Video for Asset Identification

Conventional tag-based tracking technologies are used to locate assets of significantly greater value than the tags attached to them, *e.g.* computers or personnel. Moreover, these systems conventionally use active tags whose size and thickness prevent them from being attached to small items. TRIP's almost costless resizable tags and ample range of identifiers (*i.e.* 3^9 = 19,683 different codes) features make this positioning technology suitable even for small and low-cost items, such as stationery (*e.g.* staplers or hole punchers), or books shared by personnel in an office. People will often move these items, forcing other colleagues to later search for them throughout the premises. The versatility

of TRIP to tag and locate any entity in an environment has been applied to the problem of finding the location of books shared by researchers in the LCE laboratory.

8.1.1 The LCE Sentient Library

The *LCE Sentient Library* system augments a conventional library catalogue system with a degree of context awareness. Through a web front-end a user may find out where a book was most recently seen (with an accompanying picture) and what other TRIP-tagged items were in its proximity. The functions offered through this web interface (see Figure 8.1) are: (1) browse through book categories and the books in a category, (2) perform keyword-based search of books, (3) create new book categories, (4) input the details of new books, (5) printout book-associated TRIPtags and (6) modify book details. Figure 8.2 shows the results of a book search. Note that a small cross has been overlaid on the marker representing the book sought.

At each location where a book may be left in the laboratory a *location-category* TRIPcode has been placed. Similarly, *book-category* TRIPcodes have been attached to book spines. Periodically, the LCE librarian wanders around the laboratory with a video camera recording TRIP-tagged locations and books. The Sentient Library software automatically updates the books database by the off-line processing of the video.

Ideally, the Sentient Library would use cameras spread throughout the laboratory to monitor the movements of TRIP-tagged books in real-time and so update the books database accordingly. The initial prototype, however, uses off-line processing of a video with book sightings to automatically update the catalogue, stored in the TRIP Directory Server (TDS). In fact, this approach shows the applicability of TRIP on the processing of pre-recorded video.

8.1.2 Implementation Issues

The off-line video processing nature of this application does not suit the SIF application construction model which is designed for applications requiring real-time notification of contextual information. In this case, the synchronous method interfaces offered by the respective distributed components were invoked directly. Figure 8.3 depicts the

distributed components that comprise the Sentient Library system. A Video Frame Server provides access to the serialised to disk video footage of book locations, and a TRIPparser analyses the frames captured. The TDS is used to update the contextual data associated with books as a result of book sightings. The Book Locator Agent is the core component coordinating the catalogue updating process. On initialisation, this agent uses LocALE to activate the Video Frame Server and the TRIPparser components.

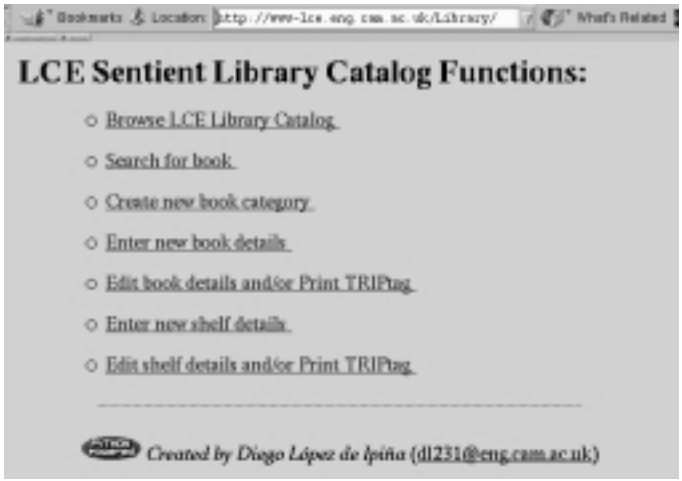


Figure 8.1. LCE Sentient Library web front-end screenshot.

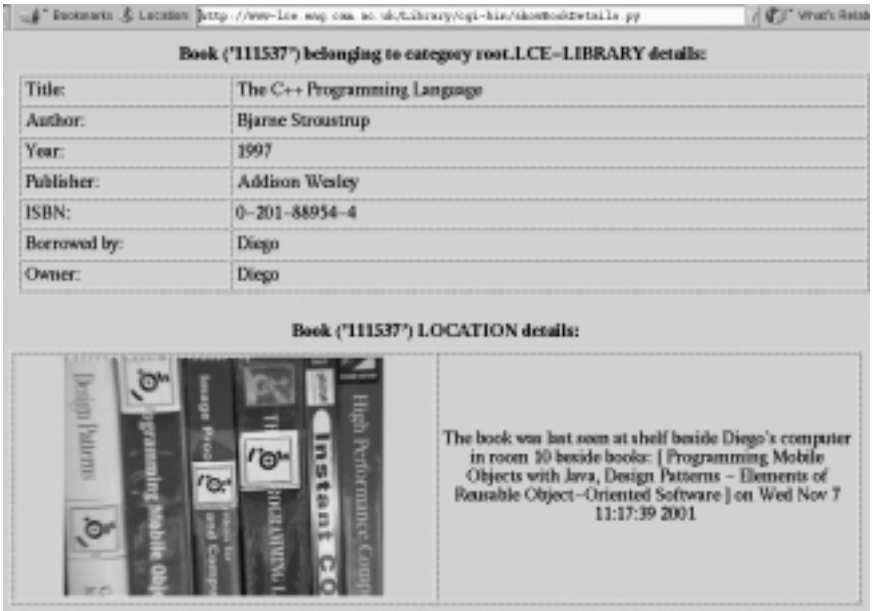


Figure 8.2. Book search results screenshot.

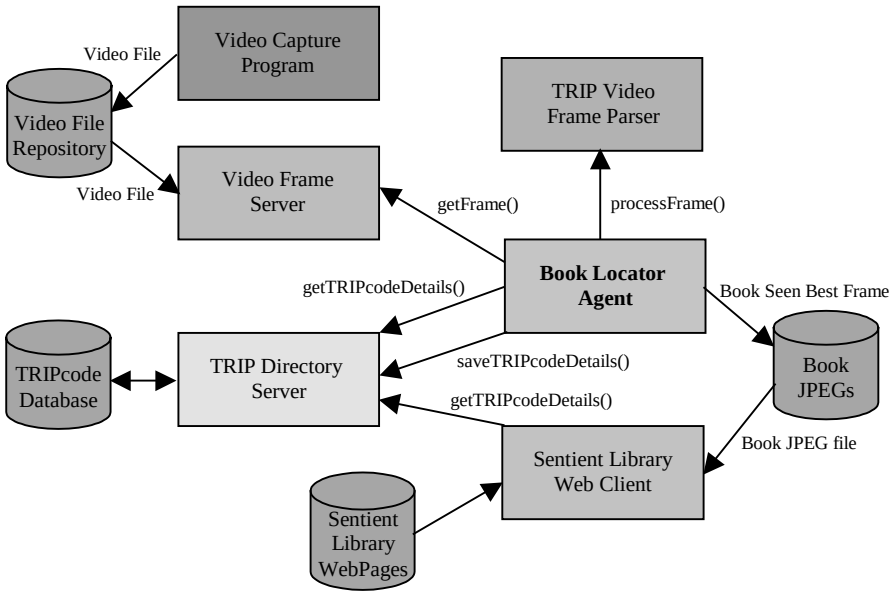


Figure 8.3. LCE Sentient Library system architecture.

The LCE Sentient Library application illustrates TRIP's versatility for tracking any tagable entities in the environment. This system has been in use for more than one year and has proven to be a popular tool in the LCE laboratory. An enhanced version of the system could be used in real libraries, where users take books from stands and later, in an attempt to 'help the librarians', place them back in the wrong locations. Librarians could be alerted of books' misplacement while scanning the book stands with a TRIP-enabled camera, rather than having to visually inspect every bookshelf in the library.

8.2 TRIP as a Device for Human Computer Interaction

The Ubiquitous Computing paradigm expounds the seamless augmentation of real-world objects with computational services. In this way, human interaction with physical objects such as desks [Wellner93] or vehicles [AT&T01b] can be enhanced. This section illustrates how the TRIP technology can be applied to augment a conventional whiteboard.

8.2.1 Active TRIPboard

The Active TRIPboard application augments a conventional whiteboard, placed in the LCE meeting room, with interactive commands issued by placing some specific TRIPcodes in the field of view of a camera focused on the whiteboard. Two example actions that can be triggered are: (1) capture whiteboard snapshot and email its web link to people currently present in the room, and (2) print a copy of the whiteboard snapshot for each person in the room. The TRIPtags used to trigger those two commands are shown in Figure 8.4. A third "reset whiteboard action" TRIPtag must be shown to the camera if the user wishes the same action to be triggered again. If a tag is not seen after a timeout, the application will reset and forget the last triggered action. This application is inspired by the BrightBoard system [Stafford-Fraser+96]. The BrightBoard system also augments a whiteboard but it uses optical character recognition (OCR) to determine the action issued (drawn) by a user, rather than the easily recognisable TRIP markers. Therefore, the BrightBoard system has higher computational demands and is less reliable, since it relies on the user's careful handwriting and knowledge of the command triggering symbols.

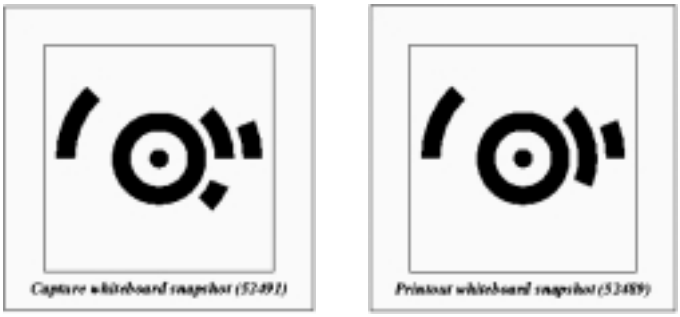


Figure 8.4. TRIPboard action control tags.

The application components: (1) a Frame Grabber that runs on the machine to which the video camera focused on the whiteboard is attached and (2) a TRIPparser that runs in any machine within the LCE laboratory's network, are activated via LocALE either when a person appears in the meeting room or through the web interface shown in Figure 8.5. The Active Badge indoor location system is used to determine the presence of people in the meeting room. Alternatively, the meeting room could have been populated with sufficient cameras to visually cover the whole space, and so permit people wearing

TRIPtags to be detected. LocALE activates the TRIPparser component by randomly picking one of the hosts in a candidate list, which achieves a degree of load-balancing. The `hostgroup` construction of LocALE's API is used to pass a list of hosts with good processing power, *i.e.* with at least 450MHz CPUs to permit a TRIP processing rate above 12 Hz. If the TRIPparser fails, say because the owner of the machine where the TRIPparser was started decides to kill that process or to reboot the machine, LocALE's fault-tolerance support recovers the component transparently to the application; LocALE will recreate the parser on another of the available hosts.

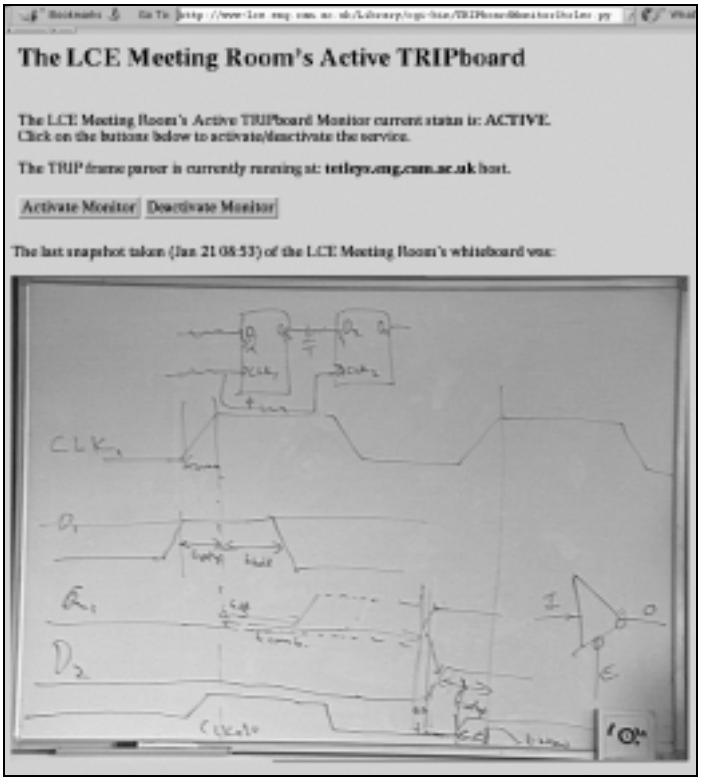


Figure 8.5. Active TRIPboard snapshot.

This application shows the use of TRIP as a device for human-computer-interaction (HCI)[21]. It also demonstrates LocALE's capability for load-balanced service activation and for transparent service recoverability. Without LocALE, this application would not

[21] The author calls this approach the Fridge Magnet Interaction Metaphor.

be fault-tolerant and its distributed components would have to be running continuously, even when the meeting room was empty.

8.3 TRIP as a Real-Time Location Sensor

Sentient spaces attempt to provide to users with the software services they expect wherever they are or move to. The goal is to allow users to move around freely without undue degradation of the computing and communications resources available to them. A common way of providing such functionality is through the so-called "Follow-Me" services. In this section, the capabilities of TRIP as a real-time identification and location sensor are used to build a *Follow-Me Audio* service.

8.3.1 Follow-Me Audio

The Follow-Me Audio service provides users with music from the nearest speakers, wherever they are. TRIPtag wearers' movements are tracked by analysing the video frames captured by web-cams installed throughout the environment. A jukebox component is controlled by showing TRIPtags representing jukebox control operations to the cameras.

Figure 8.6 shows the architecture of the Follow-Me Audio service. The core component of the system is the *Follow-Me Audio Agent*, an autonomous active object associated to a user. This agent listens to for user movement events or jukebox control actions events provided by the SIF framework, and as a result, it migrates the music with the user or it triggers operations on the digital jukebox. The Follow-Me Audio Agent operates as follows. Firstly, the agent registers its interest with the *Location Service CA*'s channel (see section 5.7) on the movement events of the user. Table 8.1 shows an example registration, expressed in the OMG's Extended Trader Constraint Language, which the agent for user 'dl231' submits to the Location Service CA's channel for the reception of this user's movement events within the LCE premises. Secondly, the agent registers, in a similar way, with the Jukebox Action Controller CA's channel. Upon arrival of the first high-level event coming from the Location Service CA, the agent activates, through LocALE, the MP3 player and jukebox components. When, it subsequently receives user movement events, the MP3 player is migrated, again using LocALE, to the computer with

audio output closest to the user. When the user moves out of the location-aware environment, the agent deactivates both the MP3 player and jukebox server. On arrival of events from the Jukebox Controller CA, the agent issues control commands to the jukebox.

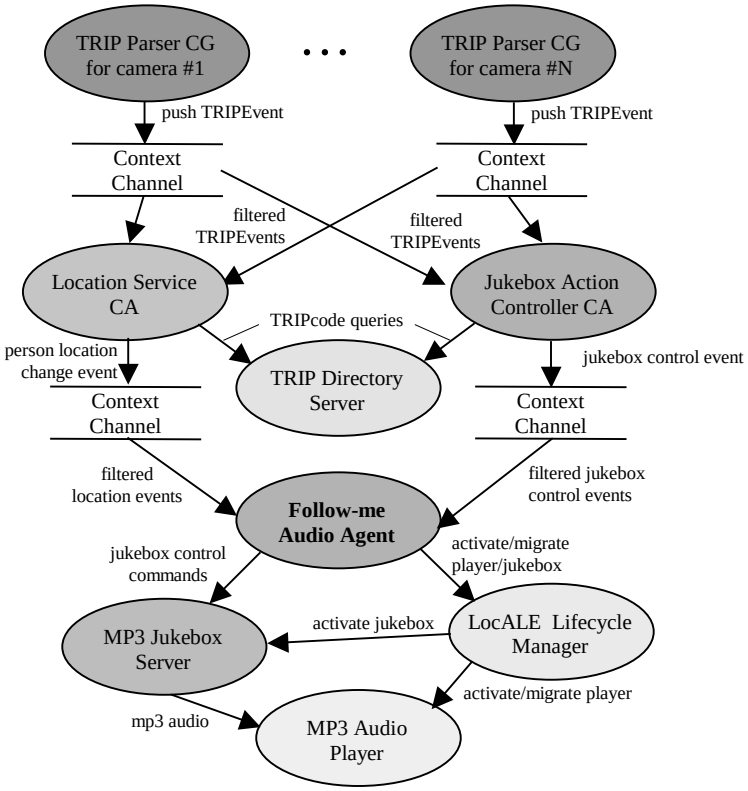


Figure 8.6. Follow-Me Audio service architecture.

The Location Service CA simply maps TRIP sighting events or Active Badge events into movement and presence events expressed in terms of regions. On the other hand, the Jukebox Action Controller Context Abstractor needs to determine when a user wearing a TRIPtag issues a jukebox control action. When both the user and the jukebox action TRIPtags are simultaneously spotted by the same camera, the Jukebox Action Controller will issue the aggregated jukebox control event understood by the Follow-Me Audio Agent. Rather than forcing the Context Abstractor to implement the monitoring of this situation, in a bespoke manner, such responsibility has been delegated to the ECA Service. For each possible jukebox action (*e.g.* play, stop, next-track and so on) that can be triggered, the CA registers with the ECA Service an ECA rule similar to the one

shown in Figure 8.7. Note that the implementation of this Context Abstractor is simplified significantly since it now only needs to first register the ECA rules corresponding to each jukebox control action and then just to handle the corresponding aggregated `Jukebox$Action` event notifications sent by the ECA Service. Both the Location Service and the Jukebox Action Controller CAs interpret the events received, with the assistance of the TDS, and produce the high-level events that can directly drive the agent operation.

Location$Movement Structured Event		
Member Name	Value Type	Description
`timestamp`	`double`	msecs since 1/1/1970
`personID`	`string`	username, e.g dl231
`from`	`string`	Format: ‘building.floor.room.region’
`to`	`string`	e.g: CUED.4.LCE-Room1.region1

```
( ( ($domain_name == ’Location’) and
    ($type_name == ’Movement’) and
    ($personID == ’dl231’) and
    (‘LCE’ ~ $to)  // is ‘LCE’ a sub-string of $.to?
  )
)
```

Table 8.1. Filtering expression on `Location$Movement` event.

```
/* RULE: Notify when user issues a jukebox "stop" action */
within 1000 { /* Ensure events occur in 1 sec time span*/
 Location$TRIPevent(cameraID ?camID, TRIPcode ?userID) and
 test("002" ~ ?userID) and /* 002 = prefix people categ. */
 Location$TRIPevent(cameraID ?camID,
                    TRIPcode "011221100") and
 /* 011221100 = stop action TRIPcode */
=>
 notifyEvent(Jukebox$Action(?userID, ?camID, "stop"));
}
```

Figure 8.7. Contextual situation specification through an ECA rule.

As users wander around the location-aware environment, the music follows them. The state of the system, including the position in the current song, persists as the components migrate. This application leverages TRIP tracking capabilities, the SIF application construction model, LocALE’s state-full object migration support and the ECA Service’s contextual event aggregation capabilities.

8.4 TRIP as a Fine-Grained Location Sensor

The three afore-mentioned TRIP-enabled applications only exploit the TRIP system's object identification capabilities. For certain kinds of location-aware applications a more fine-grained location resolution than the one typically offered by containment-based location technologies, such as the Active Badge system, is necessary. For example, the Active Badge-based desktop teleporting system [Richardson94] presents the limitation that when a user clicks her Active Badge button her desktop is teleported to one of the several screens available in a room, but not necessarily to the closest one. The user has to use one of the buttons of the Active Badge to select the desktop to teleport to, and the other button to actually initiate the teleporting. The SPIRIT [Harter+99] system solves this problem by combining the Bat's [Ward98] more precise location and orientation with resource monitoring information to allow the redirection of a user desktop to the closest (non-utilised) display. This section demonstrates how the same application can be accomplished using the much cheaper but still highly accurate and fine-grained TRIP location system.

8.4.1 TRIP-enabled Teleporting

The TRIP-enabled Teleporting service monitors when a TRIPtag-wearing user gets closer than one metre to any of the web-cams placed on top of the computers in the LCE laboratory, and those machines are not currently being used. When this situation is matched, the service automatically displays the desktop associated with that user through Virtual Network Computing (VNC) [Richardson+98], as shown in Figure 8.8. When the user later moves in front of another terminal, the VNC desktop is automatically deactivated from the previous terminal and activated on the new one.

The implementation of this service was trivial since the situation-monitoring task was delegated to the ECA Service. In the first place, the TRIP-enabled Teleport service needs to register the metadata associated with the aggregated event `Sentient$Teleport` with the ECA Service's ETR. This event type provides all the information necessary to initiate a teleport session, *i.e.* the `user` for which the service should be performed and the `host` on which it should occur. Secondly, for each host equipped with a web-cam, a rule similar to the one shown in Figure 8.9 is registered. Finally, the service simply responds

to the `Sentient$Teleport` events pushed by the ECA Service by activating the desktop for the given person at the specified host.


Figure 8.8. VNC teleport when user approaches to host “cruzcampo”.

```
/* RULE: notify when a user is spotted within 1 metre
distance of camera 0 and nobody is typing at host
"cruzcampo" where the camera is attached. */
{
 Location$TRIPevent(cameraID 0, TRIPcode ?userID,
                    d2Target ?distance) and
 test("002" ~ ?userID) and
 test(?distance < 1.0) and
 ((not query PCMonitor$keyboard_activity(host "cruzcampo",
                       intensity ?i, timestamp ?time0)) or
  (query PCMonitor$keyboard_activity(host "cruzcampo",
                    intensity ?i, timestamp ?time0) and
   test (?i < 1.0) and
   test ((curtime - ?time0) > 180000))) /* > 3 min */
=>
 notifyEvent(Sentient$Teleport(?userID, "cruzcampo"));
}
```

Figure 8.9. Contextual situation specification through an ECA rule.

The TRIP-enabled Teleport service benefits from both the fine-grained location granularity provided by the TRIP system and the situation monitoring capabilities offered by the ECA Service.

8.5 Conclusion

This chapter has demonstrated the practical value of the contributions set forth in this dissertation with the implementation of four context-aware applications. It has illustrated how the TRIP location system can process both off-line and real-time live video and be used both as an identification and fine granularity 3D location sensing technology. The ECA Service has been proven to be very useful to factor out the cumbersome composite event-monitoring task from sentient applications. Thus, sentient applications can concentrate on the implementation of the reactions required as a consequence of the matched composite or aggregated events. The LocALE middleware has been used by these applications to facilitate the deployment of their associated distributed components, enable the migration of some of these components and providing some degree of fault-tolerance. It has been shown that LocALE is a useful tool to aid the implementation of the action-dispatching aspect of sentient systems.

Chapter 9

Conclusion

This dissertation has contributed an assortment of technology solutions that cover the whole development spectrum of sentient systems, *i.e.* from context capture to context interpretation, and from context interpretation to action triggering. The aim is to foster a wider transition of sentient systems from their current successful implementation at research laboratories into homes and workplaces. Thus, user activities in those places could be enhanced and assisted by the services provided by context-aware computers. This situation has not previously been feasible given the complexity and high cost of the available location sensing technologies and the inappropriate software support provided to aid sentient systems development. The contributions put forward by this dissertation address these limitations and are deemed to make possible a more widespread adoption of Sentient Computing in more places. This chapter summarises these contributions and identifies areas that deserve further investigation.

9.1 Summary of Contributions

This dissertation has made several important contributions addressing the technology requirements necessary for making Sentient Computing more widely adopted:

1. **A real-time identification and location system**. TRIP's low-cost and easily deployable features make it suitable for locating even small assets of very low value. Images captured by low-cost, low-resolution CCD cameras are analysed through optimised image processing and computer vision algorithms in order to determine the pose of tagged objects at a rate of 16 Hz within a 3% and 2% error in 3D position and orientation, respectively.

2. **A distributed location sensor**. A CORBA-based event-driven architecture has been devised around TRIP in order to conveniently export the location information inferred by this sensor to interested parties. TRIP is, therefore, a source of contextual events, but at the same time it also acts as a sink of video frames pushed by distributed frame sources. An efficient *frame push with token feedback* protocol has been devised to enable the continuous, uninterrupted, processing of frames by TRIP, without being affected by the delay introduced by the network during frame transmission.

3. **A framework for modelling sentient applications**. The event-based model devised for TRIP has been generalised to be applicable to other types of context sources and sinks. As a consequence, three abstractions upon which sentient systems can be effectively modelled are defined. The resulting framework, termed SIF, tackles the context capture, interpretation and dissemination needs of sentient systems. Its main function is to translate context information into the forms understood by final applications.

4. **A middleware addressing the context correlation duties of sentient systems**. The ECA Service enables developers of sentient systems to efficiently separate the contextual situation monitoring from the implementation of the sentient system reaction. Developers specify rules defining contextual situations in the ECA language, submit them to the ECA Service, and implement the reactions desired when pre-specified situations are matched and notified by the service. The ECA service is very useful for the context interpretation aspect of sentient systems.

5. **A middleware addressing the action-triggering aspect of sentient systems**. The LocALE middleware represents a CORBA-based solution to the activation, migration and deactivation of heterogeneous user-associated software services. In addition, as a by-product, this framework provides fault-tolerance and load-balancing properties to the services whose lifecycle and location it controls. LocALE has proven to be a tool that simplifies the deployment of distributed components and permits the use of the spare computing resources within a network. LocALE aims to aid in the implementation of the action-triggering aspect of sentient systems.

The sentient applications described in chapter 8 have served to verify the usability of the aforementioned contributions. The development of these applications without the visual sensing and middleware support supplied would have been much more expensive and cumbersome. The combination of all of these technologies is believed to represent a solution to the aim of this dissertation, *i.e.* to make a more widespread adoption of Sentient Computing in living or working spaces a viable proposition. The contributions put forward are deemed to enable the easy integration of a Sentient Computing environment in a conventional office environment equipped with a network of hosts and some off-the-shelf web-cams.

9.2 Further Work

The TRIP location system is an operational prototype, and further work will need to be undertaken before it can be deployed as part of the infrastructure of a computing environment. Significantly, a further optimisation of the TRIP software and the adaptive operation of this system would be necessary in order to make the TRIP system more scalable. It is true that TRIP requires a low investment whilst offering fine-grained location information. However, this system's CPU processing demands are still considerable. Therefore, the scalability issues associated with the TRIP system require further investigation. On the other hand, a more efficient way of encoding information in a TRIPtag should be considered. The unique configuration for the synchronisation sector of a TRIPtag forces the use of a ternary encoding. A much greater addressing space could be achieved if a smarter quaternary encoding scheme were designed.

The scalability of the working implementation of the ECA Service should be improved. Currently, the ECA Service's event processing capabilities are limited by the characteristics of the underlying CLIPS inference engine. This engine implements the RETE many-to-many pattern-matching algorithm, which suffers from too frequent updates to the state of its knowledge base. Unfortunately, events arrive to the ECA Service very frequently and, therefore, RETE's assumption of infrequent changes to the knowledge base of the embedded engine is contravened. Algorithms with better response upon this scenario, such as RETE II [Forgy96], should be used to improve the overall performance of the ECA Service. Moreover, the elaboration of a GUI authoring tool to

create complex ECA rules in an easy manner, for even computer illiterate users, should be considered. Through a convenient GUI, users could specify the contextual situations sought and associate to them one or many commonly available actions, such as a text-to-speech processor announcing a message, an email to be sent or the display of a web page to be produced.

The LocALE middleware could be extended with certain trading service (yellow pages-like) functionality. Rather than limiting the selection of a host to run a service based on the specified location and lifecycle constraints, the LCManager could also match the resources required by the service, *e.g.* video capture, against the capabilities offered by each networked machine. In addition, the integration of LocALE with an *object location controller, e.g.* a software adaptation framework [Edmonds01], capable of deciding when and where objects should be activated or migrated to, would be interesting. In this dissertation, such intelligence has been embedded in the sentient applications developed.

Finally, the application space of the TRIP location system and the ECA and LocALE services should be expanded. Notably, the TRIP system's 3D location and orientation capabilities should be exploited for the development of applications with fine-grained location needs. For example, a TRIPtag could be used as a 3D mouse for the control of a virtual environment, *e.g.* in the Quake game. In addition, enough cameras to cover all the viewing angles of rooms in a building could be deployed in order to use TRIP in place of other indoor location technologies such as the Active Badge or Bat system. The ECA Service could be applied to assist in the development of other types of reactive systems, not necessarily falling under the Sentient Computing umbrella. Finally, LocALE could be applied to the implementation of dependable systems.

Appendix A

POSE_FROM_TRIPTAG method

Algorithm `POSE_FROM_TRIPTAG`

The input is the implicit equation of an image ellipse *C*, perspective projection of the outermost circular border of radius ρ of the bull's-eye of a TRIPtag lying on plane Π_t, and the image of the bottom outermost corner of the synchronisation sector of a TRIPtag. The focal length is set to 1, by normalising the input ellipse using the 'known' camera intrinsic parameters: $C_n = K^T \cdot C \cdot K$. The algorithm applied to C_n to obtain its location and orientation in space with regards to the camera is:

1. Compute the eigenvalues λ_1, λ_2, λ_3 of C_n ($\lambda_1 < \lambda_2 < \lambda_3$), and the corresponding eigenvectors $\vec{e_1}$, $\vec{e_2}$, $\vec{e_3}$.
2. Compute the two values $\theta = \pm \arctan \sqrt{\frac{\lambda_2 - \lambda_1}{\lambda_3 - \lambda_2}}$.
3. Compute the rotation matrix for the two possible values of θ:

$$R_C = \left[\vec{e_1} \quad \vec{e_2} \quad \vec{e_3}\right] \cdot \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

4. Set the two possible values for $\vec{n} = \left[R_{C_{13}}, R_{C_{23}}, R_{C_{33}}\right]^T$.
5. Calculate the distance from the centre of projection of the camera to the target plane:

$$d = \sqrt{\frac{-\lambda_2^2}{\lambda_1 \cdot \lambda_3}} \cdot \rho$$

6. Calculate the X''-coordinate of the centre of the imaged circle C'', δ:

$$\delta = \sqrt{\frac{-(\lambda_2 - \lambda_1)\cdot(\lambda_3 - \lambda_2)}{\lambda_1 \cdot \lambda_3}}$$

7. Calculate the two possible translation vectors joining the camera and target origins:

$$\vec{T} = R_C \cdot [\delta \quad 0 \quad d]^T$$

8. Break the ambiguity by selecting the solution that minimises the distance between the projections of point $P=[2.2\cdot r, 0, 0, 1]^T$, namely p_1 and p_2, obtained by using the two possible solutions, *i.e.* R_{C1} and $\vec{T}_1$ and R_{C2} and $\vec{T}_2$, and the known projection of this point in the image, p.

   a. Obtain the projections of P into p_1 and p_2:

$$p_i = \begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K \cdot \begin{bmatrix} R_{C_{11}} & R_{C_{12}} & T_x \\ R_{C_{21}} & R_{C_{22}} & T_y \\ R_{C_{31}}1 & R_{C_{32}} & T_z \end{bmatrix} \cdot \begin{bmatrix} 2.2\cdot r \\ 0 \\ 1 \end{bmatrix}$$

   b. Choose the solution that lies closes to p:

$$\vec{T}, n = \begin{Bmatrix} \vec{T}_1, n_1 & if & \|p_1 - p\| & < & \|p_2 - p\| \\ \vec{T}_2, n_2 & if & \|p_1 - p\| & \geq & \|p_2 - p\| \end{Bmatrix}$$

9. Calculate the real rotation matrix, R_C', by exploiting the known correspondences between points (o_c, O_c) and (x_1, X_1) in the target and image plane respectively (see Figure 4.10). Note that O_c represents the centre of the target and it is given by $\vec{T}$, whereas X_1 is the bottom outermost corner of the synchronisation sector. The back-projection of a point p_i in the image into a target point, namely P_i, is given by:

$$s_i = \frac{\overrightarrow{p_i^T} \cdot \vec{n}}{d} \;,\; \vec{P}_i = \overrightarrow{P(s_i)} = \begin{bmatrix} \overrightarrow{p_i} \\ s_i \end{bmatrix}$$

$$\vec{P}_i = \begin{bmatrix} \overrightarrow{p_c} \\ s_c \end{bmatrix} = \begin{bmatrix} p_{cx} \\ p_{cy} \\ p_{cz} \equiv 1 \\ s_c \end{bmatrix} = \begin{bmatrix} p_{cx}/s_c \\ p_{cy}/s_c \\ p_{cz}/s_c \end{bmatrix}$$

$$\vec{r}_x = \frac{(\overrightarrow{X_1} - \vec{T})}{\left\|\overrightarrow{X_1} - \vec{T}\right\|},\ \vec{r}_y = \vec{n} \times \vec{r}_x$$

$$R_C' = \left[\vec{r}_x \quad \vec{r}_y \quad \vec{n}\right]$$

10. Extract the 3 rotation angles from R_C', by applying formulae:

$$\beta = \arcsin(R_{C'_{13}}),\ \alpha = \arcsin\left(\frac{-R_{C'_{23}}}{\cos\beta}\right),\ \begin{matrix}\gamma_1 = \arcsin(-R_{C'_{12}} / \cos\beta \\ \cos\gamma_2 = R_{C'11} / \cos\beta\end{matrix}$$

If $\cos\gamma_1$ and $\cos\gamma_2$ have the same sign then γ is set to γ_1, otherwise is set to π-γ_2.

The output from the algorithm is the translation vector $\vec{T} = [T_x, T_y, T_z]^T$ and the angles (α, β ,γ) defining the 6 degrees of freedom that define the location of a target in space with respect to the viewing camera.

Appendix B

Distance Between Translation Vectors

The error incurred by choosing the wrong translation vector of the two, $\vec{T_1}$ and $\vec{T_2}$, returned by the `POSE_FROM_TRIPTAG` method is given by:

$$error = \frac{\left\| \vec{T_1} - \vec{T_2} \right\|}{\left\| \vec{T_1} \right\|} \quad \textbf{(B.1)}$$

The distance between vectors $\vec{T_1}$ and $\vec{T_2}$ and their modules are preserved when they are mapped into vectors $\vec{T_1'}$ .and $\vec{T_2'}$ by rotation R_1. The ellipse given by C' in equation (4.15) can be normalised into ellipse, E, with $a < b$, which can be used to calculate the distance between $\vec{T_1'}$ .and $\vec{T_2'}$:

$$E = \begin{bmatrix} 1/a^2 & 0 & 0 \\ 0 & 1/b^2 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad \textbf{(B.2)}$$

The assumption $a < b$ is arbitrary but if that proves to be false it is enough to switch the roles of x and y. A rotation R_2 around axis Y', identical to the one in equation (4.16), will transform E into a circle E':

$$E' = R_2^T \cdot E \cdot R_2 \quad \textbf{(B.3)}$$

From equation (B.2) and applying equations (4.19a) and (4.19b), the following expressions for the angle θ of rotation R_2 can be obtained.

$$\cos^2\theta = \frac{1+1/b^2}{1+1/a^2} \qquad \sin^2\theta = \frac{1/a^2 - 1/b^2}{1+1/a^2} \quad \textbf{(B.4)}$$

The centre of the circle *E'* for the two values of θ (see equation (4.20)), *i.e.* $\overrightarrow{T_1''}$ (for *E'* calculated using θ in R_2) *and* $\overrightarrow{T_2''}$ (for *E'* calculated using -θ in R_2), is given by:

$$\overrightarrow{T_i''} = (E')^{-1} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \textbf{(B.5)}$$

Note that the inverse of *E'* can be calculated by applying: $(E')^{-1} = R_2^T \cdot E^{-1} \cdot R_2$, where the inverse of *E* is given by:

$$E^{-1} = \begin{bmatrix} a^2 & 0 & 0 \\ 0 & b^2 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad \textbf{(B.6)}$$

Then, $\overrightarrow{T_1'}$ can be computed using -θ in R_2 and $\overrightarrow{T_2'}$ using θ in R_2:

$$\overrightarrow{T_1'} = R_2(-\theta) \cdot \overrightarrow{T_1''} = \begin{bmatrix} \sin\theta \cdot \cos^2\theta \cdot \Omega - \sin\theta \\ 0 \\ \sin^2\theta \cdot \cos\theta \cdot \Omega + \cos\theta \end{bmatrix} \quad \textbf{(B.7)}$$

$$\overrightarrow{T_2'} = R_2(\theta) \cdot \overrightarrow{T_2''} = \begin{bmatrix} -\sin\theta \cdot \cos^2\theta \cdot \Omega + \sin\theta \\ 0 \\ \sin^2\theta \cdot \cos\theta \cdot \Omega + \cos\theta \end{bmatrix} \quad \textbf{(B.8)}$$

where:

$$\Omega = b^2 \cdot (1 + \frac{1}{a^2}) \quad \textbf{(B.9)}$$

Now the distance between $\overrightarrow{T_1'}$ .and $\overrightarrow{T_2'}$ can be expressed as:

$$\left\| \overrightarrow{T_1} - \overrightarrow{T_2} \right\| = \left\| \overrightarrow{T_1'} - \overrightarrow{T_2'} \right\| = \sqrt{4 \cdot \sin^2\theta \cdot (1 - \cos^2\theta \cdot \Omega)^2} \quad \textbf{(B.10)}$$

whereas the modulus of $\overrightarrow{T_1'}$ is given by:

$$\left\| \overrightarrow{T_1'} \right\| = \sqrt{1 + \sin^2\theta \cdot \cos^2\theta \cdot \Omega^2} \quad \textbf{(B.11)}$$

In the normalised coordinate system considered, the focal length has measure 1. Supposing that the camera value of the focal length in pixels is 1,000 and the values of *a* and *b* in the original image are about 40 pixels, then in the normalised coordinate system *a* and *b* will be approximately 1/25, *i.e.* a < b << 1. Hence, considering that $\cos^2\theta \cdot \Omega = b^2 + 1$, $\Omega \approx b^2 / a^2$ and $\cos^2\theta \approx a^2 / b^2$, an expression for (B.1) can be derived:

Error incurred when choosing $\overrightarrow{T_2}$ instead of the correct vector $\overrightarrow{T_1}$

$$error = \frac{\sqrt{4 \cdot b^4 \cdot \sin^2\theta}}{1 + \frac{\sin^2\theta}{\cos^2\theta}} = \sqrt{4 \cdot b^4 \cdot \sin^2\theta \cdot \cos^2\theta} = b^2 \cdot \sin 2\theta \approx 0 \quad \textbf{(B.12)}$$

The factor $b^2 \cdot \sin 2\theta$, result of equation (B.12), is approximately *0*, as b^2 is really tiny, and the factor *sin2*θ makes it even smaller. This signifies that the error incurred by choosing the wrong $\vec{T}$ is almost negligible, *i.e.* effectively the two values obtained for $\vec{T}$ can be used interchangeably.

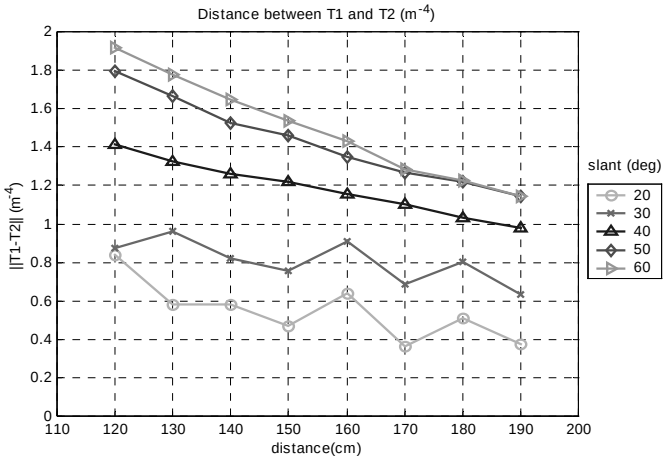


Figure B.1. Distance between $\overrightarrow{T_1}$ .and $\overrightarrow{T_2}$.

Figure B.1 depicts the Euclidean distance between the two translation vectors, *i.e.* $\overrightarrow{T_1}$ .and $\overrightarrow{T_2}$, returned by the `POSE_FROM_TRIPTAG` method in the experiments described in section 4.5.1. These results prove that both vectors are extremely close to each other. In the worse case detected $\overrightarrow{T_1}$ .and $\overrightarrow{T_2}$ lie at a distance of 0.19 mm.

Appendix C

TRIPcode Address Space Structure

A TRIPcode is composed of a prefix with the key of its category, followed by the ternary string ‘22’, and the remaining ternary digits, up to the nine the design of the TRIP targets address supports, with the target sequence number within its category. The category prefix is a ternary string with the format (xxx)*, where xxx is a three digits code in the range [000-212] and ‘*’ denotes zero or more of these sequences. For each category up to 24 subcategories can be created (range [000-212]). The range of ternary values [220-222] is reserved to denote the beginning of a valid TRIPcode. The identifier of a new subcategory is formed by adding to its parent’s category key the following non-used ternary code string in the range [000-212]. For example, the TRIPcode ‘122220121’ has as category prefix the sub-string ‘122’ and as sequence number within that category the sub-string ‘0121’.

Appendix D

The ECA Language BNF Specification

```
<rule> ::= within INTEGER <pattern_list> => <action_list> |
   <pattern_list> => <action_list>
<pattern_list> ::= <pattern_list> <connective> <pattern_list> |
   (<pattern_list>) | not <pattern_list | test(<expr>) |
   <event_pattern>
<event_pattern> ::= <EVENT_ID>(<name_value_list>) |
   <VARIABLE>:<event_pattern) | query <event_pattern>
<connective> ::= and | or | before | after
<name_value_list> ::= <name_value> |
   <name_value_list>, <name_value>
<name_value> ::= (<name_value>) | <CLASS_ID> <VARIABLE> |
   <CLASS_ID> <DATA_VALUE>
<expr> ::= <expr> and <expr> | <expr> or <expr> | not <expr> |
   <expr> <relation_op> <expr> | <expr> ~ <expr> |
   <expr> <arith_op> <expr> | - <expr> | (<expr>) |
   <DATA_VALUE> | <VARIABLE> |
   curtime | date | year | month | dayofweek
<relation_op> ::= = | <> | < | > | <= | >=
<arith_op> ::= + | - | * | /
<action_list> ::= <action_list> <action_stmt> | <action_stmt>
<action_stmt> ::= notifyEvent(<event>); |
   fireAction(<ACTION_NAME>(<params_list>)) |
   <VARIABLE> := <expr>;
<event> ::= <EVENT_ID>(<params_list>)|
   eventBatch(<event_var_list>)
<event_var_list> ::= <event_var_list>, <VARIABLE> | <VARIABLE>
<params_list> ::= <params_list>, <param_value> | <param_value>
<param_value> ::= <VARIABLE> | <DATA_VALUE>

<CLASS_ID> ::= [a-zA-Z][a-zA-Z0-9_]* (\.[a-zA-Z][a-zA-Z0-9_]*)*
<VARIABLE> ::= \?[a-zA-Z][a-zA-Z0-9_]*
<DATA_VALUE> ::= INTEGER | FLOAT | STRING
<EVENT_ID> ::= <CLASS_ID>$<CLASS_ID>
<ACTION_NAME> ::= STRING
```

Bibliography

[Abowd+01] Abowd G. D. and Mynatt E. D. “Charting Past, Present and Future Research in Ubiquitous Computing”. ACM Transactions on Computer-Human Interaction, Special issue on HCI in the new Millenium, vol. 7, no. 1, pp. 29-58, March 2001.

[Addlesee+01] Addlesee M., Curwen R., Hodges S., Newman J., Steggles P., Ward A. and Hopper A. “Implementing a Sentient Computing System”. IEEE Computer, vol. 34, no. 8, pp. 50-56, August 2001.

[Addlesee+97] Addlesee M.D., Jones A., Livesey F., and Samaria F. “The ORL Active Floor”. IEEE Personal Communications, vol. 4, no. 5, pp. 35-41, October 1997.

[Adly+97] Adly N., Steggles P. and Harter A. “SPIRIT: a Resource Database for Mobile Users”. Proceedings of the ACM CHI’97 Workshop on Ubiquitous Computing, Atlanta, March 1997.

[Andersson+01] Andersson C. “GPRS and 3G Wireless Applications”. Wiley Computer Publishing, ISBN 0-471-41405-0, 2001.

[Aramira01] Aramira Coorporation. “Jumping Beans – The Jumping Application Framework”. http://www.JumpingBeans.com, 2001.

[Ascension01] Ascension Technology Corporation. “FLOCK OF BIRDS® - Technical Description of DC Magnetic Trackers”. ftp://ftp.ascension-tech.com/pub/ascension-tech.com/Technical_Description/Theory_of_operation.doc, 2001.

[AT&T01a] AT&T Research. “omniNotify Home Page”. http://www.research.att.com/~ready/omniNotify/index.html, 2001.

[AT&T01b] AT&T Research Labs, Cambridge. “The Augmented Vehicle Project”. http://www.uk.research.att.com/augmentedvehicles/

[Azuma97] Azuma R. “A Survey of Augmented Reality”. In Presence: Tele-operators and Virtual Environments Journal, MIT Press, vol. 6, no. 4, pp. 355-385, August 1997.

[Bacon+00] Bacon J., Moody K., Bates J., Hayton R., Ma C., McNeil A., Seidel O. and Spiteri M. “Generic Support for Distributed Applications”. IEEE Computer, vol. 33, no. 3, pp. 68-76, March 2000.

[Bacon+95] Bacon J., Bates J., Richard H. and Moody K. “Using Events to Build Distributed Applications”. Proceedings of the 2nd International Workshop on Services in Distributed and Network Environments, Whistler, British Columbia, pp. 148-155, 1995.

[Bacon+97] Bacon J., Bates J. and Halls D. “Location-Oriented Multimedia”. IEEE Personal Communications, vol. 4, no. 5, pp. 48-57, October 1997.

[Bahl00] Bahl P. and Padmanabhan V. “Radar: An in-building RF-based user location and tracking system”. Proceedings of IEEE INFOCOM 2000, Tel-Aviv, Israel, pp. 775-784, March

2000.

[Barcode01] Measurement Equipment Corporation. “The Barcode Software Center”. http://www.makebarcode.com/index.html, 2001.

[Bates+96] Bates J., Halls D. and Bacon J. “A Framework to Support Mobile Users of Multimedia Applications”. ACM Mobile Networks and Nomadic Applications (NOMAD), vol. 4, no. 1, pp. 409-419, 1996.

[Bates01] Bates J. “Scaling up to a Real-Time World”, White Paper, Apama, June 2001.

[Baumann+98] Baumann J., Hohl F., Rothermel K., Schwehm M., and Straßer M. “Mole 3.0: A Middleware for Java-Based Mobile Software Agents”. Proceedings of Middleware’98, pp. 355-370, September 1998.

[Beadle+97] Beadle P., Harper B., Maguire C.Q. and Judge J. “Location Aware Mobile Computing”. Proceedings of IEEE International Conference on Telecommunications, Melbourne, Australia, April 1997.

[Beadle+98] Beadle H., Maguire G., Smith M. “Location Based Personal Mobile Computing and Communications”. Proceedings of 9th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN’98), Banff, Alberta, Canada, pp. 23-24, May 1998.

[Berk+00] Berk E. and Ananian C.S. “JLex: A Lexical Analyzer Generator for Java”, Princeton University. http://www.cs.princeton.edu/~appel/modern/java/JLex/, 2000.

[Billinghurst+99] Billinghurst, M., Kato, H. “Real World Teleconferencing”. Late-breaking paper, Proceedings of CHI '99 Extended Abstracts, ACM, pp. 194-195, May 1999.

[Bratko00] Bratko I. “Prolog Programming for Artificial Intelligence”. Longman Publishers, 3rd Edition, ISBN: 0201403757, August 2000.

[Brooks+98] Brooks R.R. and Iyengar S.S. “Multi-Sensor Fusion – Fundamentals and Applications with Software”. Prentice Hall, ISBN: 0-13-901653-8, 1998.

[Brown+01] Brown P.J. and Jones G.J.F. “Context-Aware Retrieval: Exploring a New Environment for Information Retrieval and Information Filtering”. Personal and Ubiquitous Computing Journal, Springer, vol. 5, no. 4, 2001.

[Brown+97] Brown P.J., Bovey J. D. and Chen X. “Context-Aware Applications: from the Laboratory to the Marketplace”. IEEE Personal Communications, vol. 4, no. 5, pp. 58-64, October 1997.

[Brown96] Brown P. “The Stick-E Document: a Framework for Creating Context-Aware Applications”. Proceedings of EP’96, Palo Alto, USA, pp. 259-272, January 1996.

[Brumitt+00] Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer, S. "EasyLiving: Technologies for Intelligent Environments". Proceedings of the International Conference on Handheld and Ubiquitous Computing 2000, pp. 12-27, September 2000.

[Canny86] Canny J. “A Computational Approach to Edge Detection”. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no.6, pp. 679-698, November 1986.

[Carzaniga+01] Carzaniga A., Rosenblum D. S. and Wolf A. L. “Design and evaluation of a wide-area event notification service”. ACM Transactions on Computer Systems, vol. 19, no. 3, pp. 332-

383, August 2001.

[Castro+01] Castro P., Chiu P., Kremenek T., and Muntz R. "A Probabilistic Room Location Service for Wireless Networked Environments ". Proceedings of Ubiquitous Computing 2001, Atlanta, GA, pp. 18-35, September 2001.

[Caswell+00] Caswell D. and Debaty P. “Creating Web Representations for Places”. Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K), Bristol, UK, pp. 114-116, September 2000.

[Cooperstock+97] Cooperstock J., Fels S., Buxton W., Smith K.C. “Reactive Environments: Throwing Away your Keyboard and Mouse”. Communications of the ACM, vol. 40, no. 9, pp. 65-73, September 1997.

[Dana00] Dana P.H, “Global Positioning System Overview”. The Geographer's Craft Project, Department of Geography, The University of Colorado at Boulder, http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html, May 2000.

[Daugman00] Daugman J. “How Iris Recogniton Works”. Cambridge University’s Computer Lab, Technical Report, http://www.cl.cam.ac.uk/users/jgd1000/irisrecog.pdf, 2000.

[Davies+01] Davies N., Cheverst K., Mitchell K. and Efrat A. “Using and Determining Location in a Context-Sensitive Tour Guide”. IEEE Computer, vol. 34, no. 8, pp. 35-41, August 2001.

[Davies97] Davies E. R. “Machine Vision – Theory, Algorithms and Practicalities”. Academic Press, 2nd Edition, ISBN 0-12-2060-92-X, 1997.

[Dey+00] Dey A.K. and Abowd G.D. “Towards a Better Understanding of Context and Context-Awareness”. Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness, The Hague, Netherlands, April 2000.

[Dey+99] Dey A., Futakawa M., Salber D. and Abowd G. “The Conference Assistant: Combining Context Awareness with Wearable Computing”. Proceedings of the 3rd International Symposium on Wearable Computers (ISWC’99), San Francisco, USA, October 1999.

[Dey00] Dey A.K. “Providing Architectural Support for Building Context-Aware Applications”. PhD thesis, Georgia Institute of Technology, November 2000.

[Dittrich+96] Dittrich K.R., Gatziu S. and Gepper A. “The Active Database Management System Manifesto: A Rulebase of ADBMS Features”. SIGMOD Record, September 1996.

[Edmonds01] Edmonds T. “Adaptation for Mobile Systems”. PhD Thesis, University of Cambridge, May 2001.

[ERICSSON01] ERICSSON, “Erlang Programming Language Home Page”. http://www.erlang.org/, 2001.

[eTrue01] eTrue, Inc. “TrueFace Solutions”. http://www.miros.com/solutions/face.htm, 2001.

[Farin+97] Farin G. and Hansford D. “The Geometry Toolbox for Graphics and Modelling”. A. K. Peters Publishing, ISBN 1-56881-074-1, 1997.

[Faugeras93] Faugeras O.D. “Three-Dimensional Computer Vision: a Geometric Viewpoint”. Artificial Intelligence Series. MIT Press, Cambridge, USA, ISBN 0-26206-158-9, 1993.

[Fitzgibbon+95] Fitzgibbon A.W. and Fisher R.B. “A Buyer’s Guide to Conic Fitting”. Proceedings of the 5th British Machine Vision Conference, pp. 513-522, 1995.

[Forgy82] Forgy C.L. “Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem". Artificial Intelligence, vol. 19, no. 1, pp. 17-37, 1982.

[Forgy96] Forgy C. L. “Benchmarking CLIPS/R2”. Production Systems Technologies, http://www.pst.com/bencr2.htm, June 1996.

[Forsyth+91] Forsyth D., Mundy J.L., Zisserman A., Coelho C., Heller A. and Rothwell C. “Invariant Descriptors for 3D Object Recognition and Pose”. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, no. 10, pp. 971-991, October 1991.

[Friedman-Hill01] Friedman-Hill E. “Jess: the Java Expert System Shell”. Sandia National Laboratories, http://herzberg.ca.sandia.gov/jess/, 2001.

[Gatziu+94] Gatziu S.and Dittrich K.R. “Detecting Composite Events in Active Databases using Petri Nets”. Proceedings of the 14th International Workshop on Research Issues in Data Engineering: Active Database Systems, Houston, Texas, IEEE Press, pp. 2-9, 1994.

[Glass99] Glass G. “ObjectSpace Voyager Core Package Technical Overview”. ObjectSpace, White Paper, 1999.

[González+93] González R. C. and Woods R. E. “Digital Image Processing”. Addison-Wesley Publishing Company, ISBN 0-201-60078-1, September 1993.

[Grisby99] Grisby D. P. “A Distributed Adaptive Window System”. PhD thesis, Computer Laboratory, University of Cambridge, 1999.

[Gruber+99] Gruber R. E., Krishnamurthy B. and Panagos E. “The Architecture of the READY Event Notification Service”. Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (Middleware), May 1999.

[Harter+94] Harter A. and Hopper A. “A Distributed Location System for the Active Office”, IEEE Network, pp. 62-70, Jan/Feb. 1994.

[Harter+99] Harter A., Hopper A., Steggles P., Ward A. and Webster P. “The Anatomy of a Context-Aware Application”. Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '99), Seattle, Washington, USA, August 1999.

[Hartley+00] Hartley R.I. and Zisserman A. “Multiple View Geometry in Computer Vision”. Cambridge University Press, Cambridge, UK, ISBN 0-52162-304-9, April 2000.

[Headon+01] Headon R. and Curwen R. “Recognizing Movements from the Ground Reaction Force”. Proceedings of the Workshop on Perceptive User Interfaces, Orlando, USA, November 2001.

[Henning98] Henning M. “Binding, migration and scalability in CORBA”. Communications of the ACM, vol.41, no.10, pp. 62-71, October 1998.

[Hightower+01] Hightower J. and Borriello G. “Location Systems for Ubiquitous Computing”. IEEE Computer, vol. 34, no. 8, pp. 57-66, August 2001.

[Hong+01] Hong J.I. and Landay J.A. “An Infrastructure Approach to Context-Aware Computing”. Human-Computer Interaction, vol. 16, 2001.

[Hopper00] Hopper, A., "The Clifford Paterson Lecture, 1999 Sentient Computing". Philosophical Transactions of the Royal Society London, vol. 358, no. 1773, pp. 2349-2358, August 2000.

[Huang+99] Huang A.C., Ling B.C., Ponnekanti S. and Fox A. “Pervasive Computing: What Is It Good For?”. Workshop on Mobile Data Management (MobiDE), MobiCom '99, Seattle, WA, September 1999.

[Hudson99] Hudson S. “CUP Parser Generator for Java”, Princeton University, http://www.cs.princeton.edu/~appel/modern/java/CUP/, 1999.

[Hull+97] Hull R., Neaves P. and Bedford-Roberts J. “Towards Situated Computing”. Proceedings of the International Symposium on Wearable Computers, Boston, pp. 146-153, October 1997.

[Hylton+97] Hylton J. and Van Rossum G. “Using the Knowbot Operating Environment in a Wide-Area Network”. 3rd ECOOP Workshop on Mobile Object Systems, Jyväskylä, Finland, June 1997.

[IBM01] IBM Corporation, “Amit – Active Middleware Technology”. http://www.haifa.il.ibm.com/projects/software/amit/, 2001.

[Ipiña+01a] López de Ipiña D. and Lo S. “LocALE: a Location-Aware Lifecycle Environment for Ubiquitous Computing”, ICOIN-15, Beppu, Japan, February 2001.

[Ipiña+01c] López de Ipiña D. and Lo S.L. "Sentient Computing for Everyone". Proceedings of the 3rd International Conference on Distributed Applications and Interoperable Systems (DAIS'2001), Krakow, Poland, September 2001.

[Ipiña+01d] López de Ipiña D. and Katsiri E. "An ECA Rule-Matching Service for Simpler Development of Reactive Applications". Published as a supplement to the Proceedings of Middleware 2001 at IEEE Distributed Systems Online, vol. 2, no. 7, November 2001.

[Ipiña00] López de Ipiña D. "Building Components for a Distributed Sentient Framework with Python and CORBA". Proceedings of the 8th International Python Conference, Arlington, VA, USA. January 2000.

[Ipiña01b] López de Ipiña D. "Video-Based Sensing for Wide Deployment of Sentient Spaces". Proceedings of 2nd PACT 2001 Workshop on Ubiquitous Computing and Communications, Barcelona, Catalonia, Spain, September 2001.

[Iridian01] Iridian Technologies, “Iridian Technologies Home Page”. http://www.iridiantech.com/, 2001.

[Jackson96] Jackson I. “Anonymous Addresses and Confidentiality of Location”. Proceedings of Information Hiding: 1st International Workshop, Springer, Cambridge, U.K., pp. 115-120, June 1996.

[Kato+99] Kato H. and Billinghurst M. “Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System”. Proceedings of the 2nd Internation Workshop on Augmented Reality, pp. 85-94, October 1999.

[Kindberg+00] Kindberg T., Barton J., Morgan J., Becker G., Caswell D., Debaty P., Gopal G., Frid M., Krishnan V., Morris H., Schettino J., Serra B., and Spasojevic M. “People, Places, Things: Web Presence for the Real World”. 3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA2000), December 2000.

[Kirsch+97] Kirsch D. and Staner T. “The Locust Swarm: An environmentally-powered, network-less location and messaging system”. Proceedings of the 1st International Symposium on Wearable Computers, pp. 169-170, October 1997.

[Korkea-aho00] Korkea-aho, M. “Context-Aware Applications Survey”. Internetworking Seminar (Tik-110.551), Helsinki University of Technology, 2000.

[Kortuem+98] Kortuem G., Segall Z., Bauer M. “Context-Aware, Adaptive Wearable Computers as Remote Interfaces to ‘Intelligent’ Environments”. Proceedings of the 2nd International Symposium on Wearable Computers, 1998.

[Koschel+98] Koschel A. and Kramer R. “Configurable Event Triggered Services for CORBA-based Systems”. *Proceedings of the 2nd International Enterprise Distributed Object Computing Workshop (EDOC'98)*, pp. 306 – 318, November 1998.

[Kruizinga01] Kruizinga.P. “The Face Recognition Home Page”. Department of Computing Science, University of Groningen, The Netherlands, http://www.cs.rug.nl/~peterkr/FACE/face.html, 2000.

[Krumm+00] Krumm J., Harris S., Meyers B., Brumitt B., Hale M. and Shafer S. “Multi-Camera Multi-Person Tracking for EasyLiving”. Proceedings of 3rd International Conference on Visual Surveillance, Dublin, Ireland, pp. 3-10, July 2000.

[Lamming+94] Lamming M. and Flynn M. “Forget-Me-Not - Intimate Computing in Support of Human Memory”. Proceedings of FRIEND21, ’94 International Symposium on Next Generation Human Interfaces, Japan, February 1994.

[Leonhardt98] Leonhardt U. “Supporting Location-Awareness in Open Distributed Systems”. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, May 1998.

[Liu+99] Liu G. and Mok A.K. “Composite Events for Network Event Correlation”. Proceedings of IFIP/IEEE International Symposium on Integrated Network Management, May 1999.

[Lo+00] Lo S.L., Riddoch D. and Grisby D. “The omniORB version 3.0 user's Guide”, AT&T Laboratories Cambridge http://www.uk.research.att.com/omniORB/doc/3.0/omniORB/index.html, May 2000.

[Long+96] Long S., Kooper R., Abowd G. D., and Atkeson C. G. “Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study”. Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking (MobiCom'96), pp. 97-107, November 1996.

[Maffeis96] Maffeis S. “The Object Group Design Pattern”. Proceedings of the USENIX 1996, Conference on Object-Oriented Technologies (COOTS), Toronto, Ontario, Canada, June 1996.

[McCandless97] McCandless M. “Managing Your Privacy in an Online World”. IEEE Expert, pp. 76-77, January/February 1997.

[Microsoft01] Microsoft Corporation. “Microsoft .NET Home Page”. http://www.microsoft.com/net/default.asp, 2001

[Microsoft96] Microsoft Corporation. “DCOM Technical Overview”. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp, November 1996.

[Mills92] Mills D.L. “Network Time Protocol (Version 3) Specification, Implementation and Analysis”. RFC 1305, Network Working Group, University of Delaware, http://www.faqs.org/rfcs/rfc1305.html, March 1992.

[MIT01] “Things That Think Web Site”, http://www.media.mit.edu/ttt/, 2001

[Mitchell+00] Mitchell S., Spiteri M.D., Bates J. and Coulouris G. “Context-Aware Multimedia Computing in the Intelligent Hospital. Proceedings of Ninth ACM SIGOPS European Workshop, Kolding, Denmark, September 2000.

[Moran+99] Moran T.P., Saund E., van Melle W., Gujar A.U., Fishkin K.P. and Harrison B.L. “Design and technology for Collaborage: collaborative collages of information on physical walls”. Proceedings of the 12th annual ACM symposium on User interface software and technology, November 1999.

[Naguib+01] Naguib H., Coulouris G. and Mitchell S. “Middleware Support for Context-Aware Multimedia Applications”. Proceedings of DAIS 2001, pp. 9-22, Kluwer Academic Publishers, September 2001.

[NASA99] NASA, “CLIPS: A Tool for Building Expert Systems”. http://www.ghg.net/clips/CLIPS.html, August 99.

[OMG00] Object Management Group. “Life Cycle Service Specification”. ftp://ftp.omg.org/pub/docs/formal/00-06-18.pdf, April 2000.

[OMG00a] Object Management Group, “Trading Object Service Specification”. ftp://ftp.omg.org/pub/docs/formal/00-06-27.pdf, May 2000.

[OMG00b] Object Management Group, “Notification Service Specification”. ftp://ftp.omg.org/pub/docs/formal/00-06-20.pdf, June 2000.

[OMG01] Object Management Group, “The Common Object Request Broker Architecture: Architecture and Specification. Revision 2.5”. ftp://ftp.omg.org/pub/docs/formal/01-09-01.pdf, September 2001.

[OMG01a] Object Management Group, “Naming Service Specification”. ftp://ftp.omg.org/pub/docs/formal/01-02-65.pdf, February 2001.

[OMG01b] Object Management Group, “CORBA/IIOP Specification, Chapter 25, Fault Tolerant CORBA”. http://www.omg.org/cgi-bin/doc?formal/01-09-01, September 2001.

[OMG01c] Object Management Group, “CORBAservices Specifications”. http://www.omg.org/technology/documents/spec_catalog.htm#CORBAservices, October 2001.

[Orr+00] Orr R. and Abowd G. D. “The Smart Floor: A mechanism for natural user identification and tracking”. Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI 2000), The Hague, Netherlands, April 2000.

[Parker97] Parker J.R. “Algorithms for Image Processing and Computer Vision”. Wiley Computer Publishing, ISBN 0-471-14056-2, 1997.

[Parthus01] Parthus Corporation. “Parthus Launches and Licenses NavStream 3000 GPS Platform Enabling Location Determination in the Most Challenging of Indoor and Urban Canyon Environments”. http://www.parthus.com/news/press_releases/october_09_2001.html, October 2001.

[Pascoe98] Pascoe J. “Adding generic contextual capabilities to Wearable Computers”. Proceedings of the Second International Symposium on Wearable Computers, Pittsburgh, USA, October 1998.

[Paton+99] Paton N.W. and Díaz O. “Active Databases Survey”. ACM Computing Surveys, vol. 31, no.1, pp. 63-103, March 1999.

[Pilu+96] Pilu M., Fitzgibbon A., Fisher R. “Ellipse-specific Direct least-square Fitting ". IEEE International Conference on Image Processing, September 1996.

[Pollefeys00] Pollefeys M. “3D Modeling from Images”. Tutorial Notes, ECCV 2000, Dublin, Ireland, June 2000.

[Pryantha+00] Pryantha N.B., Chakraborty A. and Balakrishnan H. “The Cricket location-support system”. Proceedings of the Sixth Annual ACM International Conference on Mobile Computing and Networking (MOBICOM00), Boston, MA, August 2000.

[Randell+01] Randell C. and Muller H. “Low Cost Indoor Positioning System”. Proceedings of Ubicomp 2001, Atlanta, USA, pp. 42-48, September 2001.

[Real01] Real Technologies “Net Eye Gold CCTV product”. http://www.realtechnologies.co.uk/neteyegold/neteyegold_moreinfo.html, 2001.

[Rekimoto+00] Rekimoto J. and Ayatsuka Y. “CyberCode: Designing Augmented Reality Environments with Visual Tags”. Proceedings of Designing Augmented Reality Environments (DARE), April 2000.

[Rekimoto+95] Rekimoto J. and Nagao K. “The World Through the Computer: Computer Augmented Interaction with Real World Environments”. Proceedings of User Interface Software and Technology (UIST’95), 1995.

[Rekimoto98] Rekimoto J. “Matrix: a Real-Time Object Identification and Registration Method for Augmented Reality”. Proceedings of Asia Pacific Computer Human Interaction (APCHI’98), July 1998.

[Richardson+98] Richardson T., Stafford-Fraser Q., Wood K.R. and Hopper A. “Virtual Network Computing”. IEEE Internet Computing, vol.2, no.1, pp 33-38, Jan/Feb 1998.

[Richardson94] Richardson T. “Teleporting in an X Window System Environment”. IEEE Personal Communications Magazine, vol. 1, no. 3, pp. 6-12, 1994.

[Rizzo+94] Rizzo M, Linington P.F., and Utting I.A. “Integration of Location Services in the Open Distributed Office”. Technical Report 14-94, Computing Laboratory, University of Kent, Canterbury, UK, August 1994.

[Salber+99] Salber D., Anind K. and Abowd G.D. “The Context Toolkit: Aiding the Development of Context-Enabled Applications”. Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), Pittsburgh, PA, pp. 434-441, May 1999.

[Samani+97] Mansouri-Samani M., M. Sloman. “GEM: A Generalised Event Monitoring Language for Distributed Systems”. IEE/BCS/IOP Distributed Systems Engineering, vol. 4, no. 2, pp. 96-108, June 1997.

[Schilit+94a] Schilit B.N., Adams N.L. and Want R. “Context-Aware Computing Applications”.

In Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, USA, December 1994.

[Schilit+94b] Schilit B. and Theimer M. “Disseminating Active Map Information to Mobile Hosts”, IEEE Network, vol. 8, no. 5, pp. 22-32, September/October 1994.

[Schmidt+97] Schmidt D.C. and Vinoski S. “Object Adapters: Concepts and Terminology”. SIGS C++ Report, vol. 11, November 1997.

[Schmidt+98] Schmidt D. and Vinoski S. “C++ Servant Managers for the Portable Object Adapter”. SIGS C++ Report, September 1998.

[Schmidt+99] Schmidt A., Beigl M. and Gellersen H. “There is more to Context than Location”. Computers & Graphics Journal, Elsevier, vol. 23, no.6, pp 893-902, December 1999.

[Sonka+99] Sonka M., Hlavac V. and Boyle R. “Image Processing, Analysis, and Machine Vision”. PWS Publishing, ISBN 0-534-95393-X, 1999.

[SONY01] “SONY VAIO C1 PictureBook™”. http://www.sonystyle.com/vaio/picturebook/index.html, 2001.

[Spreitzer+93] Spreitzer M. and Theimer M. “Providing Location Information in a Ubiquitous Computing Environment”. Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, Asheville, USA, December 1993.

[Stafford-Fraser+96] Stafford-Fraser J.Q. and Robinson P. “BrightBoard: A Video-Augmented Environment”. Proceedings of Conference on Human Factors in Computing Systems (CHI96), April 1996.

[Stafford-Fraser96] Stafford-Fraser J.Q. “Video-Augmented Environments”. PhD thesis, University of Cambridge, February 1996.

[Stillman+99] Stillman S., Tanawongsuwan R. and Essa I. “A System for Tracking and Recognizing Multiple People with Multiple Cameras”. Proceedings of Second International Conference on Audio-Vision-based Person Authentication, Washington, DC, April 1999.

[SUN01] Sun Microsystems, “Java™ 2 Platform, Enterprise Edition (J2EE) Home Page”. http://java.sun.com/j2ee/, June 2001.

[SUN02] Sun Microsystems, **“**JavaTM 2 SDK, Standard Edition, v 1.4.0 Release Home Page”. http://java.sun.com/j2se/1.4/, January 2002.

[SUN97] Sun Microsystems, “Java Native Interface Specification 1.1”. http://java.sun.com/products/jdk/1.2/docs/guide/jni/spec/jniTOC.doc.html, May 1997.

[SUN98] Sun Microsystems, “Java Object Serialization Specification”. ftp://ftp.javasoft.com/docs/jdk1.2/serial-spec-JDK1.2.pdf, November 1998.

[SUN99] Sun Microsystems, “Java Remote Method Invocation Specification v. 1.3”. ftp://ftp.java.sun.com/docs/j2se1.3/rmi-spec-1.3.pdf, December 1999.

[Tekinay98] Tekinay S. “Wireless Geolocation Systems and Services”. Special Issue of the IEEE Communications Magazine, vol. 36, no. 4, April 1998.

[Thomas+97] Thomas G. A., Jin J., Niblett T. and Urquhart C. “A versatile camera position

measurement system for virtual reality in TV production". Proceedings of IBC'97, pp. 284-289, September 1997.

[TIBCO00] TIBCO Software Inc. "TIBCO InConcert". http://www.tibco.com/products/in_concert/, November 2000.

[Trimble01] "GPS Tutorial", 2001, http://www.trimble.com/gps/, 2001.

[Trucco+98] Trucco E. and Verri A. "Introductory techniques for 3D Computer Vision". Prentice-Hall, Inc., ISBN-0-13-261108-2, 1998.

[Tsai87] Tsai R. Y. "A versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses". IEEE Journal of Robotics and Automation, vol. RA-3, no. 4, pp. 323-344, August 1987. Source code available at: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/rgw/www/TsaiCode.html

[UDDI01] UDDI.org, "Universal Description, Discovery and Integration of Services for the Web Home Page". http://www.uddi.org/, October 2001.

[Visionics01] Visionics Corp. "FaceIt: Face Recognition Technology". http://www.visionics.com/faceit, 2001.

[W3C00] The World Wide Web Consortium, "Simple Object Access Protocol (SOAP) 1.1". http://www.w3.org/TR/SOAP/, May 2000.

[W3C01] The World Wide Web Consortium, "Web Services Description Language (WSDL) 1.1". http://www.w3.org/TR/wsdl, March 2001.

[Want+01] Want R. and Schilit B. "Expanding the Horizons of Location Aware Computing – Guest Editors' Introduction". IEEE Computer, Special issue on Location-Aware Computing, vol. 34, no. 8, pp. 31-34, August 2001.

[Want+92] Want R., Hopper A., Falcão A. and Gibbons J. "The Active Badge Location System". ACM Transactions on Information Systems, vol. 10, no. 1, pp. 91-102, January 1992.

[Want+95] Want R., Schilit B., Adams N., Gold R., Goldberg D., Petersen K., Ellis J., Weiser M. "An Overview of the PARCTab Ubiquitous Computing Experiment". IEEE Personal Communications, vol. 2, no. 6, pp. 28-43, December 1995.

[Ward98] Ward A. "Sensor-driven Computing". PhD thesis, Cambridge University Engineering Department, UK, August 1998.

[Weatherall02] Weatherall J. "An Embedded Ubiquitous Control Architecture for Low Power Systems". PhD Thesis, Cambridge University Engineering Department, February 2002.

[Weiser92] Weiser M. "The Computer for the 21st Century". Scientific American, vol. 265, no. 3, pp. 94-104, September 1992.

[Wellner93] Wellner P. "Interacting with paper on the DigitalDesk". Communications of the ACM, vol. 36, no. 7, pp. 87-96, August 1993.

[Werb+98] Werb J. and Lanzl C. "Designing a positioning system for finding things and people indoors". IEEE Spectrum, pp.71-78, September 1998.

[Wren+97] Wren C., Azarbayejani A., Darrell T. and Pentland A. "Pfinder: Real-Time tracking of

the Human Body". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, July 1997.

[Zhang00] Zhang Z. "A flexible new technique for camera calibration". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 11, pp. 1330-1334, 2000.