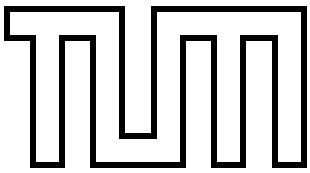
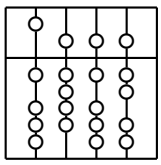


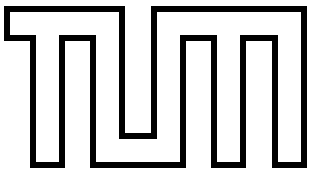
Technische Universität
München
Fakultät für Informatik



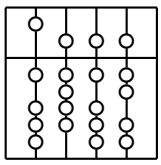
Diplomarbeit

# Bootstrapping of Sensor Networks in Ubiquitous Tracking Environments

Franz Strasser



Technische Universität
München
Fakultät für Informatik



Diplomarbeit

# Bootstrapping of Sensor Networks in Ubiquitous Tracking Environments

Franz Strasser

Aufgabensteller: Univ-Prof. Gudrun Klinker, Ph.D.

Betreuer: Dipl.-Inf. Martin Wagner

Abgabedatum: 15. Juli 2004

## Erklärung

Ich versichere, dass ich diese Ausarbeitung der Diplomarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Juli 2004 Franz Strasser

## Zusammenfassung

Anwendungen aus dem Bereich Augmented Reality (AR, Erweiterte Realität) verschmelzen die reale und virtuelle Welt. Positionsbestimmung (Tracking) von realen Objekten ist Hauptaufgabe jeder AR Anwendung, weil sie deren räumlichen Zusammenhänge kennen muss, um mit dem Benutzer zu interagieren. Anwendungen aus dem Bereich Ubiquitous Computing (Ubicomp, „Allgegenwärtige Computerumgebungen“) versuchen intelligente Geräte in die Umgebung zu integrieren. Durch die Kombination der Konzepte von Ubiquitous Computing und der Erweiterten Realität können Trackingtechnologien dynamisch in neue Anwendungen eingebaut werden. *Ubiquitous Tracking* („Allgegenwärtiges Tracking“) verbindet verschiedene Tracker zu einem Sensorennetzwerk, in dem alle verschiedenen Typen von mobilen und stationären Sensoren zu dem Gesamtsystem beitragen.

Diese Diplomarbeit beschäftigt sich mit der Integration von mobilen Trackern (Klienten) mit anderen stationären und mobilen Trackern. Der Klient besitzt kein Vorabwissen über seine umgebende Infrastruktur. Der Schwerpunkt dieser Arbeit ist, ein konzeptionelles, komponenten-basiertes Modell zu finden, das geeignet ist, mobile Klienten in groß angelegte Sensorennetzwerke einzubinden. Um das in einer skalierbaren Art und Weise zu bewerkstelligen, muss das Sensorennetzwerk in kleinere, selbst-verwaltbare Teile aufgeteilt werden, z.B. durch Einführung eines hierarchischen Ortskontextes.

Ich werde die Anforderungen an ein Allgegenwärtiges Trackingsystem analysieren und eine komplette Übersicht über alle möglichen Szenarien geben, in denen zwei getrennte Systeme zur Laufzeit verbunden werden. Die Annahme ist, dass wenigstens eines davon ein mobiler Klient ist, der kein Wissen von dem anderen System hat. Durch den Austausch von Konfigurationsinformationen können beide Systeme ihre Hardwareausstattung (um-)konfigurieren, um gegenseitiges Tracking zu ermöglichen. Falls diese Konfiguration nicht vollständig das komplette Sensorennetzwerk beschreiben kann, können die Sensoren lediglich unbekannte Objekte ausmachen.

Eines der größeren Probleme ist, diese unbekannte Objekte mittels ihrer eigenen Bewegungsmuster wieder zu erkennen und zu identifizieren. Durch das Untersuchen der linearen Geschwindigkeit und der Winkelgeschwindigkeit von zwei Objekten können statische Beziehungen zwischen ihnen erkannt werden, ohne eine wirkliche Messung von Sensoren zu haben. Die Hauptidee ist, Zusammenhänge in den beiden Werten durch Frequenzanalysen zu finden.

Ich werde Ergebnisse der Untersuchungen an verschieden Trackingtechnologien präsentieren, wie z.B. das ART Trackingsystem, ARToolKit und Intersense. Sie werden zeigen, inwieweit diese Sensoren für die Frequenzanalysen geeignet sind.

**Abstract**

Augmented Reality (AR) applications fuse the real and the virtual world together. Tracking of real objects is a major part of any AR application because it must know their spatial relationships to interact with the user. Ubiquitous Computing applications try to integrate intelligent devices into the environment. By combining concepts of Ubiquitous Computing and Augmented Reality, tracking technologies can be dynamically integrated in new applications. *Ubiquitous Tracking* aggregates different trackers into one sensor network where all different types of mobile and stationary sensors contribute to the whole system.

This thesis deals with the integration of mobile tracking set-ups (clients) into stationary or mobile ones. The client itself does not have any initial knowledge of its ambient infrastructure. The focus of this thesis is to find a conceptual component-based model for dynamically integrating the mobile clients into large-scale sensor networks. In order to accomplish this in a scalable way, the network has to be divided into smaller, self-manageable parts, e.g. by introducing a hierarchical location context.

I analyze the requirements for a Ubiquitous Tracking system and give a complete overview over all possible scenarios where two separated systems get connected at runtime. The assumption is that at least one of them is a mobile set-up which has no initial knowledge of the other. By exchanging configuration, information the two systems can (re-)configure their hardware equipment to enable the tracking of each other. If this information does not describe the whole sensor network completely, the sensors can merely track unknown objects.

One of the major issues is to recognize and identify these unknown objects by their own motion patterns. By investigating the speed and angular velocity of two objects from two different trackers, static relationships between them can be discovered without an actual measurement of a sensor. The main idea is to find correlation by analyzing the frequencies of those two values.

I present results of the investigation of different tracking technologies, such as the ART tracking system, ARToolKit and Intersense. They show how far these sensors are suitable for the frequency analysis.

# Preface

## Purpose of This Document

This thesis was written as Diplomarbeit, which is adequate to a Master's thesis, at the Technische Universität München's Chair of Applied Software Engineering. From January through July 2004, two other Computer Science Master's students and I designed and developed a first distributed implementation of Ubiquitous Tracking, a new concept for a unified tracking model. My thesis is one result of this joint project, called Ubitrack.

In this thesis, I would like to explain the ideas behind my work, document its results, and show its future implications.

## Target Audience

This thesis addresses several different audiences because the chapters focuses on different topics. This section provides an overview of all chapters and who should read them.

**General Readers** who are not familiar with the research area of Augmented Reality should read chapter 1. For a good introduction into the formal concepts of Ubiquitous Tracking and DWARF, they should read chapter 2 to get an overview of the project.

**Augmented Reality Researchers and other Computer Scientists** should start with chapter 3 which introduces into the focus of my thesis in detail. But if they are interested in the formal model of Ubiquitous Computing, they should start reading at chapter 2. Chapter 4 presents simple scenarios which I investigated. Chapters 6 and 7 show the details about estimating static spatial relationships from deduced tracking data, a problem found after in the scenarios.

**Dwarf Developers** might read chapters 2 and 5 for details. They contain the needed information for the favored implementation. Chapter 8 could provide further ideas for development.

**Future Ubitrack Developers** should read the chapters 2, 4, 6 and 7. These chapters provide the best overview of all of my work solving very specific problems.

## Timeline

The order of the chapters reflects the rough timeline of my diploma thesis.

Chapter 1 shows all key concepts of Augmented Reality and Ubiquitous Computing with which I got familiar between November 2003 until January 2004.

Chapter 2 contains the overall design of our implementation which we worked on between January and February 2004.

Chapters 3 and 4 reflect an analysis phase to identify problems and find solutions for a general bootstrapping model of our implementation design. This phase took part between February and March.

Chapter 5 shows implementation concepts for parts of the bootstrapping model. Work on this chapter started in March and lasted until April.

Chapters 6 and 7 presents solutions and results for a very specific problem generated from the scenarios. This was the last phase and took place between April and June 2004.

The last chapter concludes all chapters by listing the lessons I learned while writing the thesis. It also lists issues which can be considered as future work.

## Electronic Version

If you are a DWARF developer and you are currently reading a paper version of this thesis, make sure to get hold of the electronic version, in Portable Document Format (PDF).

## Acknowledgments

This thesis would never have been possible to be written without the restless efforts on many people.

First I want to thank the other members of the project: Dagmar Beyer and Daniel Pustka. Without them my muddleheaded ideas would become even more weird for myself. Thanks for clarifing them.

Secondly I want to thank Gudrun Klinker. Without her, the door to Augmented Reality would never be opened. She pointed out very interesting ideas.

Special thanks go to my supervisors, Martin Wagner, Asa MacWilliams and Martin Bauer, for their enormous efforts for supporting us over a long period of time for the project. This was incredible.

Very special thanks go to all my friends for supporting and proofreading, especially for Mario Gleirscher (for his motivating tea sessions), Peter Weiss and all other "Gabelsbergler".

Finally, I would like to thank my family, who always helped me to keep my feet on the ground: *Ohne Euch hätte ich es nie geschafft!*

Garching, July 2004

Franz Strasser

I would very much like to hear any comments or suggestions you may have regarding this thesis or my overall work. This especially includes any questions you may have. Please do not hesitate to contact me at franz@streeter.de.

# Overview

# Contents

# List of Figures

# 1 Introduction

**Introduction to Augmented Reality and Ubiquitous Computing; motivation for Ubiquitous Tracking and Ubitrack; goals of my work.**

---

This chapter gives an overview of the research field of Augmented Reality. It presents the goals of my diploma thesis, while readers who are not familiar with this field should read it for an introduction.

## 1.1 Motivation of Augmented Reality

Before I explain the term "Augmented Reality", I present the motivations of the people who work in this research area. This short introduction should create an intuitive understanding of this research field.

The development of computers allowed people to perform complex calculation at a fraction of time compared with humans. According to Moore's law [41], computing power will double every 18 months. This trend continues as current figures from Intel Research [31] prove. As computing power evolves, new application areas are discovered. They do not only perform calculations anymore: they print newspapers, enhance and analyze pictures and even control cars. They influence the real world and thereby influence us. Computers are fast data processors and deal with information faster than any human is capable to. Therefore humans delegate data processing to the machines. Their produced results are then evaluated. Since devices are getting more and more important for humans they have to get more integrated into everyday life. Therefore interaction with humans is getting more important. Though there is a discrepancy between how the human user deals with information and how the computer does. This discrepancy depends from the points of view of a user and a machine:

### 1.1.1 The User's Point of View

We use computers to perform complicated tasks that would not be possible without fast calculation power of electronic devices. Although we use them, we do not want to "see" them – apart from the hardware gurus of course. Normally we want to interact with them in a natural way.

Donald A. Norman [43] uses the term "invisible computer" to describe the attribute common to all effective, easy to use and popular tools in human culture: no matter how complicated the functions they may perform, the technology used to perform an action is not apparent to the user of the tool. Such tools are human-centered, not technology centered. The technology is invisible. The contact with computers should only be explicit for the users when they want to put in information or want feedback.

**Natural Input** Communication is an essential part of our life. In order to exchange information between individuals, humans rely among other things on visual, audible, and tactile interactions. We want to interact in the same natural way even with electronic devices. However, these devices have very restricted mechanisms for communication which are far from being natural. We still use keyboards and mice to communicate with desktop PCs. Using speech as input medium becomes more relevant since it is faster and more natural compared to typing texts or clicking on icons.

**Natural Feedback** But we do not only want to use natural input methods; we want to see reactions to our requests: Has our request been understood by the machine? What are the results? Results must be processed by the application so that we can understand them.

The most natural way is to present the results of the input by integrating it into the ambient real world. For example, the prices for products in any shops are printed next to them and not on a list at the cash desk. The information is provided when needed and where needed.

### 1.1.2 The "Device's" Point of View

These mechanisms are far from being ideal for machines. How humans see and interpret the world is totally different from the perspective of a machine. Computer applications provide solutions for certain problems for the user. They are, however, bound to the specific computer system. Information is modeled by binary numbers and by applying arithmetic operations on them, results are generated. Computers cannot deal with other information except numbers; and pure numbers are the least intuitive representation of information for humans.

### 1.1.3 Connecting the Real and Virtual World

This is the discrepancy: on the one hand the human world, the real world; and on the other hand the computer's world, the virtual world. Therefore models must be found to map the information from the user's real world to the virtual world of numbers in a machine. One research area which is dedicated to solving the conflict of mapping the user's world into the computer's world and vice versa is Augmented Reality.

## 1.2 What is Augmented Reality?

*Augmented Reality* (AR) fuses the real and the virtual world together while trying to find solutions to interact naturally with computers. The name itself already describes the main purpose: augmenting the real world, i.e. enhancing the world with information generated by computers. This can be necessary since information is not always available when it is needed. AR tries to solve this problem to offer techniques for combining the two worlds together.

Probably everyone is aware of the term *Virtual Reality* (VR). This term stands for a completely computer generated world where users can influence a virtual scene. In contrast to AR it tries to generate a complete and realistic model of the real world whereas AR uses the real world itself. The virtual world is completely independent from the "outside" world.

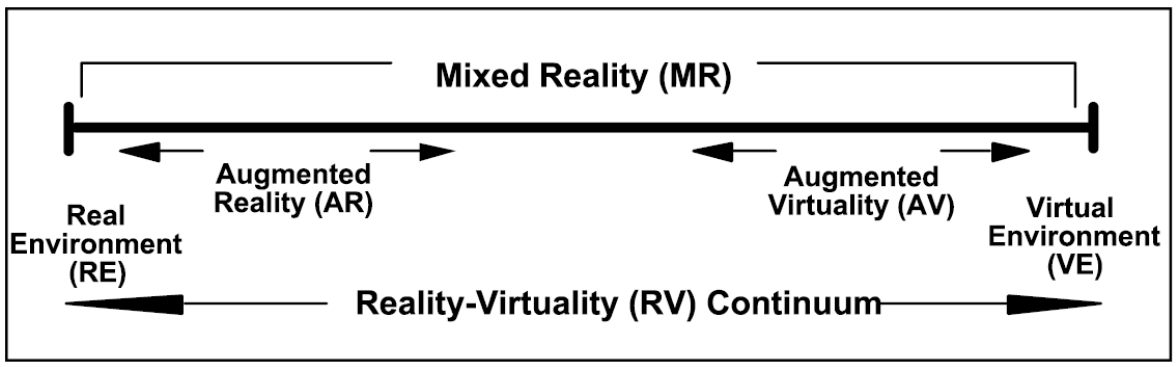



**Figure 1.1:** The Mixed Reality continuum (Paul Milgram and Herman Colquhoun Jr. [40])

AR and VR are combined under the common expression *Mixed Reality* [40]. Mixed Reality is the continuum which lies between the real environment and the virtual environment (figure 1.1). As you can see you can consider VR as the opposite research field.

Augmented Reality integrates virtual generated information into the real world by using special kinds of technologies. Classical AR applications use *head-mounted displays* (HMDs) or *head-up displays* (HUDs) using specific display techniques like *optical see-through* or *video see-through*. According to [3] AR applications are determined by three characteristics:

1. AR combines the real and virtual world
2. User are allowed to interact with AR applications in real-time
3. AR is registered in 3-D, i.e. all objects (real or virtual) have a determined spatial position

Possible applications range from indoor use [2, 52] to outdoor applications [5]. These classical AR set-ups in general are monolithic and designed for exactly one application. Furthermore, they are self-contained including the used technology.

Despite these classical AR applications where computer generated information is augmented into the environment by using special display technologies, other research fields use alternative approaches. In contrast to the application-centered view of classical AR where the information is augmented by the user's application, you can enhance the real world by embedding intelligent devices in the environment. Computers therefore become ubiquitous.

## 1.3 Ubiquitous Computing

Embedding devices into the user's environment and rendering them invisible is another approach for interaction between humans and computers. Since hardware becomes cheaper, wearable, and wireless, *Ubiquitous Computing* [59] (Ubicomp) integrates intelligent devices in wearable tools. For example, PDAs assists the user in organizational tasks, mobile phones allow new degrees of connectivity. Therefore, when analyzing the development of computer technology over the last century, an important trend can be found out: over the years the relation of how many people use how many computing devices changed (figure 1.3).


**Figure 1.2:** Pictures showing typical Augmented Reality applications

As mentioned, Ubicomp goes a different way than Augmented Reality to support users with computer generated information. They are surrounded by computers which are embedded into the environment or into intelligent tools. Typically one cannot see the devices at all. The main characteristic of Ubicomp environments is that the different components are distributed with no central coordination component. Every device serves for a very special purpose and by combining them they build a specific application. Since these devices are embedded in the environment the above mentioned invisibility of hardware is realized. So they become part of every day life. An interesting analogy can be seen in the development of the Internet:

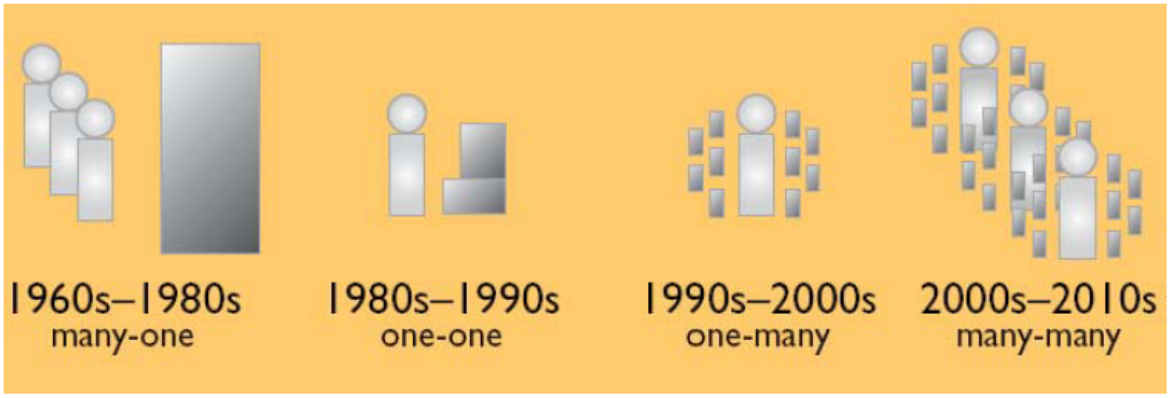
> The concept of computers as things that you walk up to, sit in front of and turn on will go away. In fact, our goal is to make the computer disappear. We are moving towards a model we think of as a "personal information cloud". That cloud has already begun to coalesce in the form of the Internet. The Internet is the big event of the decade [...]. We'll spend the next 10 years making the Net work as it should, making it ubiquitous.
>
> *Frank Casanova, director of Apple Computer Inc.'s Advanced Prototyping Lab*

Typical application areas of Ubiquitous Computing are *intelligent buildings*. Nearly everything – from coffee machines to sun blinds – are controlled by invisible embedded computers. Research is done by several institutes, including e.g. Microsoft Research [12].

Ubicomp can be seen as a natural addition to Augmented Reality. Since both approaches come from different directions, combining them will create new possibilities in both research fields.

### 1.3.1 Combining Augmented Reality and Ubiquitous Computing

The fusion of the research fields of Augmented Reality and Ubiquitous Computing seems natural, since both try to enhance the real world. AR enhances it with concepts of VR,




| Years | Paradigm | What happened there? |
|---|---|---|
| 1960 – 1980 | Mainframe Computing | Traditionally, computer systems where designed that many users can work with one device, the mainframe. This was due to the fact that hardware was expensive and therefore only central systems were profitable. |
| 1980 – 1990 | Personal Computing | While hardware became cheaper, every one could afford a Personal Computer. So one user worked with one device at a time. |
| 1990 – 2000 | Mobile Computing | Because hardware became smaller and networks wireless the development of wearable intelligent devices allowed people to use many devices at the same time. However in general they are dedicated to only one specific person and/or one specific application. |
| 2000 – 2010 | Ubiquitous Computing | This is the current development: small, even invisible devices enrich the users' environment. They can be used collaborative at the same time for different applications. |

**Figure 1.3:** Evolution from computer mainframes to Ubicomp environments (taken from Roel Vertegaal, Attentive User Interfaces [57])

whereas Ubicomp enhances the real world by embedding intelligent machines in the environment. As mentioned in [27]:

> Ubiquitous Computing [...] and Augmented Reality each have their own merits. When combined together they can help realize "the empowerment of the user during human-computer interaction, as well as the embodiment of a sense of control". "Feeling in control" is a key concept in the context of the oncoming information society. People should be able to get the information they need (and deserve to get) in a convenient way. They should not feel intimidated or dominated by either the amount of information coming towards them, or by the technology (e.g. computers or computing devices) needed to get it.

The mentioned "feeling in control" is the key word in this consideration. Although many devices are contributing to the user's work, not every device can be directly controlled. Therefore, the application behavior has to adapt to the user's current behavior. This adaptiveness can be realized by generating and using external context information.

### 1.3.2 Context-Awareness

An important keyword in Ubicomp is *context*. Since the user cannot control all devices at the same time, they have to estimate for themselves what the users are currently doing and what they probably want to do. The context can be used to automatically react to actions of the user. A general definition of context is given by A. K. Dey and G .D. Abowd of Georgia Institute of Technology:

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

Context is getting important when Ubicomp devices have to interact with users automatically without their direct control. Context-aware applications should be perceptive in respect to the user: they should react as the user expects them. In order to be able, the application must collect context information. A possible taxonomy of these context information can be informally expressed by the following questions:

**Identity:** Who is the user?

**Time:** When did an action happen, or when should it happen?

**Activity:** What did the user do, or what does he wants to do?

**Location:** Where is the user?

Context-aware applications may use all these information to react automatically on input. A simple example is that the lights should be switched on when the user enters a room and dimmed to his/her preferences.

### 1.3.3 Sensor Networks for Context Derivation

The context information is derived by the embedded sensing devices in the environment. They can determine all kind of quantities, from noise level to temperature to position. In general these sensors are small devices which must consume few energy. The communication is typically wireless. Since these devices are small and may not provide direct access after they are installed, the communication must be dynamic and robust. An overview of sensor networks can be found in [26, 28, 21].

The *location context* becomes an important part of the design when Ubicomp meets AR. Location context is deduced by *sensors* which measure the positions of *locatables*. On the hardware layer sensors deliver spatial information in form of 3-D position and rotation data. To gather location context information the sensors must form networks to exchange their data. By fusing and integrating the information, high-level context information can be generated. Architectures for context derivation can be found in [15, 17].

## 1.4 Tracking in Augmented Reality

After the introduction into the main concepts of Ubiquitous Computing, this section introduces into the determination of spatial information in AR. In contrast to the wide variety of different sensors in Ubicomp the basic sensor type in classical AR applications are *trackers*. This section presents only an overview of the tracking area. It should be considered as an introductory section for a very basic understanding of tracking and the underlying tracking technology. For a more complete overview, see [1].

Augmented Reality uses the term *tracking* for the process of determination the position and/or orientation of an object in 3-D space. This is because of the fact that Augmented Reality applications are registered in 3-D space and therefore need spatial information. Typically every tracking technology has its own frame of reference in order to estimate the values. *Position* is the the 3-D vector from the origin to the object in the coordinate system of the frame. *Orientation* expresses how the object is rotated relative to the axes of the coordinate system.

In contrast to the above mentioned sensor networks, Augmented Reality tracking does not have restriction of wireless communication and low-energy consumption of the hardware. This is due to the fact that classical tracking set-ups are not mobile and designed only for a specific AR application.

### 1.4.1 Mathematical Background

I will give a very short introduction of the mathematical methods and representations which are used in the tracking area. The use of linear algebra and numerical mathematics is essential in this area. However, it is not as difficult as it looks like at first glance.

#### Position

As already mentioned, position is the 3-D point in the tracker's coordinate system where the object resides. Typically it is expressed as a 3-D vector containing the values which are estimated by the tracking technology.

Most common is the homogeneous representation of this vector with four components instead of three: $\mathbf{v} = (x, y, z, w)^T$ where $x$, $y$, and $z$ are the coordinates and typically $w = 1$.

#### Orientation

Orientation is bit more complicated since there are several representations which have all advantages and trade-offs.

**Rotation Matrices** A rotation matrix is used to rotate a set of points within a coordinate system. While the individual points are assigned new coordinates, their relative distances do not change. All rotations are defined using the trigonometric "sine" and "cosine" functions. For 3-D there are three rotation matrices for each axis. They look as follows:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \qquad R_y = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

where $\alpha$ is the angle of the rotation around the axis. Orientation can be expressed by the matrix multiplication of these rotation matrices $R_x(\alpha) * R_y(\beta) * R_z(\gamma)$.

**Euler angles** Euler angles are the name given to the set of rotation angles which specify the rotation in each of the X, Y and Z rotation axis. These are specified in vector format $(x, y, z)$.

**Quaternions** Quaternions are a non-commutative extension of the complex numbers. They may be represented by 4-dimensional vector $q = w + xi + yj + zk$ where $w$, $x$, $y$ and $z \in \mathbb{R}$. $i$, $j$ and $k$ are the imaginary units with $i^2 = j^2 = k^2 = ijk = -1$. You can also write $q$ as $q = (w, \mathbf{v})$ with $\mathbf{v} = (x, y, z)$.

According to [20], every rotation can be expressed by a quaternion of the form

$$q = \begin{pmatrix} \cos\frac{\alpha}{2} \\ \mathbf{v}\sin\frac{\alpha}{2} \end{pmatrix}$$

where $\mathbf{v}$ specifies the rotation axis and $\alpha$ the angle.

### 1.4.2 Different Tracking Technologies

Besides the mathematical background, I want to introduce a very short list of tracking technologies which I used in my thesis. This section is only intended to give a review of the used technologies. More detailed and complete presentations can be found in [3] or [1].


**Figure 1.4:** Examples of ARToolKit markers used in Ubitrack: They may contain figures, text or other patterns.

### ARToolKit

ARToolKit [33, 32] is a vision-based tracking system developed at the Osaka University, Japan, and is being supported by the Human Interface Technology Laboratory (HIT Lab.) at the University of Washington, and the HIT Lab NZ at the University of Canterbury, New Zealand.

ARToolKit extracts features from video images and computes the camera position relatively to detected quadratic patterns in the video stream that we call *marker* (figure 1.4). All markers have the same geometric structure. They are quadratic with an arbitrary size $s$ and have a black border with thickness $\frac{s}{4}$. The interior consists of a pattern which has the size $\frac{s}{2} \cdot \frac{s}{2}$. The pattern has to be generated and must be loaded into the detection subsystem. If a pattern is recognized by the detection subsystem, a positions reconstruction subsystem calculates the relative position of the camera to the marker.

### ART tracking system

The ART tracking system[1] is an optical tracking system based on two or more infrared sensitive cameras. They are mounted in a room and cover a certain cubic area. These cameras are equipped with CCD image sensors, working in the near infrared light spectrum. An infrared light flash illuminates the measurement volume periodically. Little reflecting spheres are combined to *targets* (how the ART locatables are called). The reflection of the spheres are recorded by the CCD chips in the camera which calculate the 2-D position in the camera image. These positions are transfered to a computer running a software that computes the 3-D position in the room. Figure 1.5 shows one camera and one target.

### Intersense Intertrax

Intersense's Intertrax is a mobile inertial tracker. In principal it measures the acceleration of the tracker itself and delivers the rotation in form of yaw-pitch-roll triples.

[1]The ART tracking system is a commercial product of Advanced Realtime Tracking GmbH, Herrsching bei München (`http://www.ar-tracking.de/`)

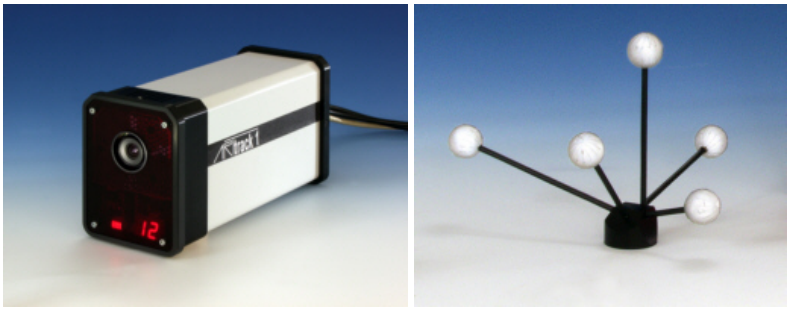


**Figure 1.5:** Picture of an ART camera and target

## 1.5 Introduction to Ubiquitous Tracking

Tracking is an essential basis of whole AR applications. Without tracking many of the natural interaction techniques would not be possible. However tracking techniques in general are bound to a specific area. If concepts of Ubiquitous Computing and Augmented Reality are combined, tracking technologies have to become part of ubiquitous sensor networks.

> *This part of my diploma thesis was jointly written and published by the members of the Ubitrack project at the Technische Universität München: Dagmar Beyer [10], Daniel Pustka [48] and Franz Strasser.*

### 1.5.1 Definition

Augmented Reality is a natural interface to Ubiquitous Computing environments. However, classical AR application are limited to the range of their tracking systems. When they use wide-area trackers, the update rate and accuracy of the measurements are low and quite often do not fulfill the requirements of AR. Moreover, they assume that sensors are deployed statically in the area of interest and that these sensors are pre-calibrated off-line. These drawbacks inhibit the development of mobile, large-scale, and dynamic systems as they were found in Ubicomp. Every tracking method also suffers from various disadvantages due to their used technology.

*Ubiquitous Tracking* combines available tracking technologies to one large sensor network. It provides unified APIs for applications and transparently delivers measurement results. The data is fused automatically and dynamically from heterogeneous sources.

### 1.5.2 Motivation for Ubiquitous Tracking

Every tracking technology suffers from various disadvantages: inertial trackers provide only relative measurements and suffer from drift, video-based camera systems need lots of computing power for image processing and the accuracy in the line of sight is rather low. Sensors also have a limited range where they can act. However, Ubiquitous Computing environment are typically large-scaled with many sensors.

The weaknesses of the different technologies are compensated by using different sensors at the same time. For example, camera-based methods can be paired with inertial tracker to compensate the drift. By adding GPS you even will get a global frame of reference.

To extend the range of sensors, *sensor fusion* combines the measurements and fuses them together. You can combine stationary indoor tracking systems with GPS in order to extend their operation range. Significant and pioneering work has be done, amongst others, by Azuma [4], et al. (see related work 2.5). However, no attempt has been made by them to automate the integration of sensors into these large-scale sensor networks.

Classic AR applications use static tracking platforms designed exactly for that one purpose. So reusing tracking components is hard to achieve. Ubiquitous Tracking also tries to abstract from tracking hardware by introducing a reusable, component-based model of data flow and processing.

### 1.5.3 Goals of Ubiquitous Computing

The goal is to obtain an optimal estimate of arbitrary geometric relationships and their accuracy at any time for a user-specified definition of optimality. Such tracker abstraction can help applications handle the varying levels of tracker uncertainty that affects the registration of virtual objects in 3-D space.

The system itself is designed to allow dynamic cooperation between distinct and mobile components. Changes in the infrastructure, i.e. adding new hardware or occurring failures, are recognized whereupon the system tries to minimize the error or even to deliver better results. The developer is shielded from these influences by a software layer which offers a simple, efficient API to the developer.

### 1.5.4 Issues which arise

These goals however imply problems which have to be addressed:

1. The most obvious issue is the incompatibility between the various outputs of tracking technologies. For example, one sensor sends its orientation information as rotation matrix, the other one as Euler angles. One sensor delivers position and orientation whereas the other only accelerations.

2. These measurements vary in their accuracy. A generic error model has to be found to specify all possible measurements errors which may occur.

3. All objects which should be 3D-registered in the application have to be modeled and identified. If new objects are tracked, the model must be updated on the fly.

4. Since Ubiquitous Tracking abstracts from the hardware, it must deal with changes of the infrastructure at runtime. If hardware fails, the application should not be aware of that fact.

To address these issues a new project called *Ubitrack* was started.

## 1.6 The Ubitrack Project

*This part of my diploma thesis was jointly written and published by the members of the Ubitrack project at the Technische Universität München: Dagmar Beyer [10], Daniel Pustka [48] and Franz Strasser.*

In the winter term 2003/04 the Augmented Reality group of the Applied Software Engineering chair at the Department of Informatics, Technische Universität München offered diploma theses on the topic *Ubiquitous Tracking*. Three students, Dagmar Beyer, Daniel Pustka, and Franz Strasser started to work on an implementation of Ubiquitous Tracking. The AR group in Munich and the Vienna University of Technology, Austria developed a formal Ubiquitous Tracking model [42]. The new joint project between these universities was named *Ubitrack*.

The overall purpose of the three diploma theses in Munich was to find an implementation of Ubitrack which should be usable in Augmented Reality applications but should be based on a distributed, decentralized Ubicomp architecture at the same time. In the following months we developed a DWARF system model of Ubitrack, mapped it onto new and existing components of DWARF, implemented a demo set-up, and evaluated our own theses with this demo. The detailed concept and models are explained in chapter 2.

### 1.6.1 Design goals

To define the design goals and the API, a workshop was held from February, 6th till February, 8th 2004 at Entrudisalm near Salzburg, Austria. The resulting design goals are:

**Transparency** Ubitrack should be framework which should provide a query mechanism for the applications. So the framework should shield the application from the tracking hardware as much as possible.

**Scalability** Since AR applications are enriched with concepts of Ubicomp, the desire for large-scale tracking networks is growing. Such networks must be scalable in respect of the size of available hardware and infrastructure because current intelligent environments may contain thousands of sensors.

**Flexibility** Ubitrack must be a very flexible framework in respect to changes in the infrastructure. New hardware should be easily integrated at runtime and changes in the communication network should be considered.

### 1.6.2 Layered Architecture

Transparency is achieved by introducing a three layer architecture model. The application is shielded from the hardware by introducing an intermediate *Ubitrack layer* (figure 1.6). Between these layers there are defined APIs which allow the application and the hardware to communicate with Ubitrack.

All parts of the application that want to communicate with Ubitrack lie in the application layer. The *Query API* is the interface between the layers and defines how queries have to be structured and how results are delivered.

The hardware layer contains all components which deliver positional informations about objects. They range from sensors to databases where static relationships are stored, e.g. static relationships of walls in a building. These low-level results are aggregated in the Ubitrack layer and delivered according to the query to the application. The *Sensor API* how low-level results are delivered to the Ubitrack layer.

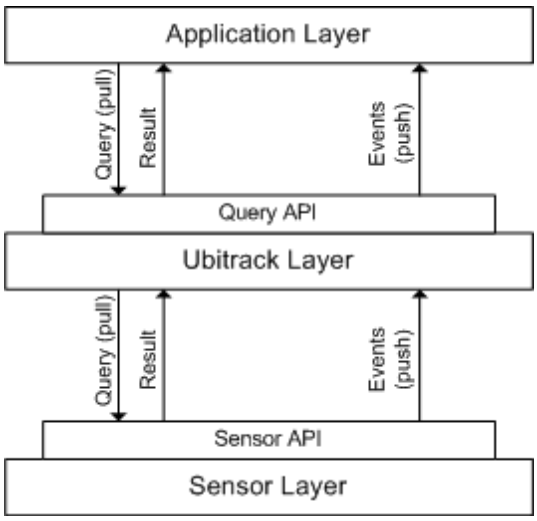



**Figure 1.6:** The three layer Ubitrack architecture model

## 1.7 Goals of my work

My overall goal is the integration of separated mobile Ubitrack set-ups in other set-ups. This means that Ubitrack tries to combine tracking mechanism on the fly to form large sensor networks dynamically. This should happen, without manual influence of the user and automatically by the Ubitrack system.

Therefore I present concepts for a bootstrapping mechanism for Ubitrack tracking components. My proposed solutions exchange configuration data in order to set up the tracking devices for the new environment. I restrict myself on cases where a mobile set-up should be integrated in a stationary or another mobile set-up.

My personal involvement covers three main areas:

1. Reusing existing DWARF concepts to develop a model how hardware sensors and locatables can describe themselves and how these descriptions can be deployed and propagated to the needed locations. These concepts have to be adapted to Ubitrack framework.

2. Finding initial knowledge of the infrastructure to exchange configuration information for the sensor hardware.

3. Finding new spatial relationship in order to combine the two separated systems either by exchanging hardware descriptions to (re)configure the sensors and their corresponding software or by analyzing common motion patterns.

4. Using motion based analysis for estimating static spatial relationships where no actual measurements are available.

5. Presenting results of the analysis of motion patterns to determine if new spatial relationships can be estimated without actual measurements.

Parts 1–3 are discussed broadly since they cover open research fields which cannot be covered in detail in this thesis but are relevant for Ubitrack. Part 4 and 5 present results of an

offline simulation and analysis to verify the proposed solutions. I also give ideas how this can be implemented in the current DWARF Ubitrack system.

# 2 Ubiquitous Tracking and DWARF

**Description of the formal model of Ubiquitous Tracking; basics of DWARF; the first DWARF-based implementation.**

---

This chapter is an introduction to the basics of Ubitrack. First, it presents the formal model of the concepts for Ubiquitous Tracking. Then, it gives a short overview of the basics of DWARF, an AR framework for developing distributed applications. Thirdly, it will present a simple example which shows the key concepts of a DWARF-based Ubitrack implementation. These concepts are then describes in detail. The chapter concludes with related work.

> *This part of my diploma thesis was jointly written and published by the members of the Ubitrack project at the Technische Universität München: Dagmar Beyer [10], Daniel Pustka [48] and Franz Strasser.*

## 2.1 Formal Model

First, the chapter will present the formal mathematical model for Ubiquitous Tracking.

### 2.1.1 Spatial Relationship Graph

Within an Augmented Reality environment, it is essential to maintain an awareness of its occupying objects and their physical relationship to each other. In respect to the system's flexibility and scalability it must be possible to configure the setup of the environment dynamically and to extend it easily. The goal of the formal model discussed in this section is to provide a method to obtain optimal estimations of the spatial relationships between arbitrary objects within an Augmented Reality environment. For this purpose a coherent, up-to-date spatial model of the environment is essential.

A suitable data structure to store the spatial model is a directed graph. Nodes in this *Spatial Relationship Graph* (SR Graph) represent locatable objects, which could be either active sensors such as cameras or passive objects such as markers or other targets. The directed edges represent the spatial relationships between two objects, this means the position and orientation of the object represented by the target node relative to the object represented by the source node.

#### Real Relationships

From a general point of view, for each pair of objects a spatial relationship can be determined for each point in time. This spatial relationship can be expressed by a transformation of the coordinate frame of the source object to the coordinate frame of the target object, for

example by using the computer graphics notation of a $4 \times 4$ homogeneous matrix, a $3D$ translation vector and a quaternion, or any alternative representation. In the following, we will represent the transformation by using a $4 \times 4$ homogeneous matrix.

In figure 2.1 an example for such a Spatial Relationship Graph is given which contains three objects $A$, $B$, and $C$.

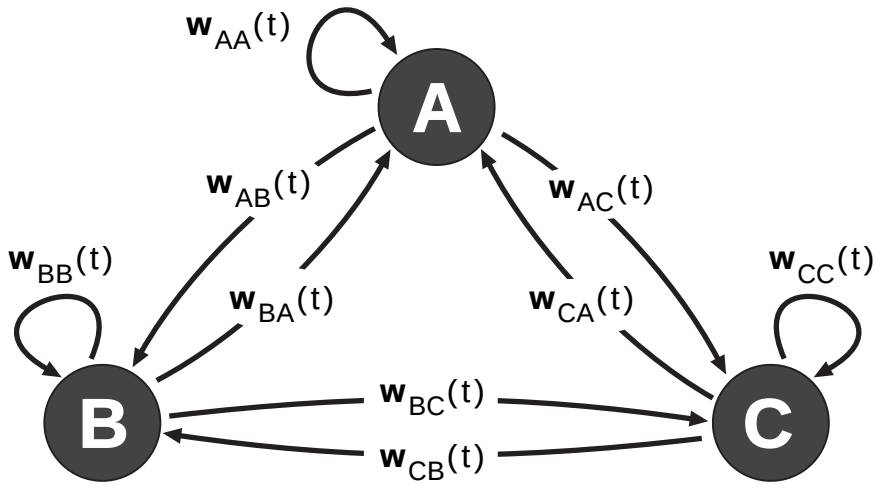



**Figure 2.1:** Spatial Relationship Graph according to relation $\Omega$, showing the real spatial relationships between objects

Defining a binary relation $\Omega$ on the object space $N$ (in our example, $N = \{A, B, C\}$), each element $(X, Y) \in \Omega$ can be mapped onto a function $w_{XY}$ describing the transformation of the coordinate frame of $X$ to the coordinate frame of $Y$ over time. This attribution scheme is called $\mathbf{W}$.

$$\mathbf{W} : (\Omega = N \times N) \rightarrow w, \text{ where } w : D_t \rightarrow \mathbb{R}^{4\times4} \tag{2.1}$$

Each directed edge in our example Spatial Relationship Graph is annotated by a function $w$ that maps the time domain $D_t$ onto the spatial relationship domain $\mathbb{R}^{4\times4}$. In this complete graph, all spatial relationships between arbitrary objects can be obtained at each point in time.

#### Measured Relationships

Needless to say, that it is not possible to have such an omniscient view on the environment's underlying geometry as described above. Therefore in the real world, only estimations of spatial relationships exist on the basis of measurements taken at a discrete point in time.

The measurements representing the real spatial relationships are corrupted by noise and systematic errors depending on the accuracy provided by the tracking devices. In order to take this varying quality of the estimations into account, a set of attributes is added to each measurement, which describes properties such as the standard deviation of the positional information or latency between the actual measurement and the time this measurement is delivered. In section 2.1.2 a more detailed view on this attributes is given.

In order to extend the formal model to this further considerations, a relation $\Phi$ and an attribution scheme $\mathbf{P}$ is defined:

$$\mathbf{P} : (\Phi \subseteq N \times N) \rightarrow p, \text{ where } p : D_t \rightarrow \mathbb{R}^{4\times4} \times \mathcal{A} \tag{2.2}$$

A directed edge between two objects $X$ and $Y$ exists only if measurements have been taken. Every discrete time point when a measurement is taken is mapped by the function $p_{XY}$ to a homogeneous matrix representing the measured spatial relationship and a set of attributes describing the quality of this measurement. An example for a Spatial Relationship Graph according to relation $\Phi$ is given in figure 2.2.

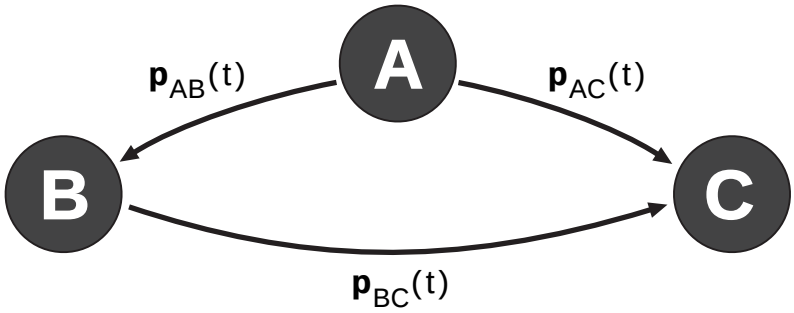



**Figure 2.2:** Spatial Relationship Graph according to relation $\Phi$, showing the measured spatial relationships between objects

Assuming that in the example the geometric relation between $A$ and $B$ was measured twice at time $t_1$ and $t_2$, this leads to two homogeneous matrices $H_1$ and $H_2$ with the according sets of attributes $\mathcal{A}_1$ and $\mathcal{A}_2$. Hence, function $p_{AB}$ is defined as:

$$\begin{array}{rcccl} p_{AB} & : & \{t_1, t_2\} & \rightarrow & \mathbb{R}^{4\times4} \times \mathcal{A} \\ & & t_1 & \mapsto & (H_1, \mathcal{A}_1) \\ & & t_2 & \mapsto & (H_2, \mathcal{A}_2) \end{array} \tag{2.3}$$

#### Inferred Relationships

This estimation of spatial relationships is in many ways insufficient. Due to the limited range and the specific properties of tracking devices, it is not possible to provide information about spatial relationships between all objects. As measurements are performed at discrete points in time, it is also impossible to provide estimations of the spatial relationships continuously. Therefore it is very unlikely, that an application's request for a specific spatial relationship can be fulfilled by the measured relationships. It becomes necessary to extend the knowledge of the environment's geometry by inferring new estimations for spatial relationships on the basis of measured relationships. Accordingly, the attributes describing the quality of the inferred spatial relationships must be adjusted to reflect the loss of quality in respect to the quality of measured relationships. In order to provide positional information about objects at arbitrary points in time, interpolation or extrapolation functions can be used, or for a more accurate prediction, Kalman Filter or particle filters. Inverse edges can be added to the Spatial Relationship Graph by inverting the measured relationships. Estimations of spatial relationships at the same point in time can be fused along paths in the Spatial Relationship

Graph in order to form the transitive closure of the graph. Just as measured relationships are provided by different tracking software components, each inferred relationship is provided by a new software component performing this inference on the measured positional information. For a more detailed description of these new software components, see section 2.1.3.

All these possible forms of inferring new spatial relationships lead to the final goal to define a binary relation $\Psi$ that approximates the idealized relation $\Omega$ in the real world:

$$\mathbf{Q} : (\Psi \subseteq N \times N) \rightarrow Q, \text{ where } Q = \{q \in Q \mid q : D_t \rightarrow \mathbb{R}^{4\times4} \times \mathcal{A}\} \tag{2.4}$$

Each element $(X, Y) \in \Psi$ is mapped onto a set of functions $q_{XY}$, which are describing specific forms of inference.

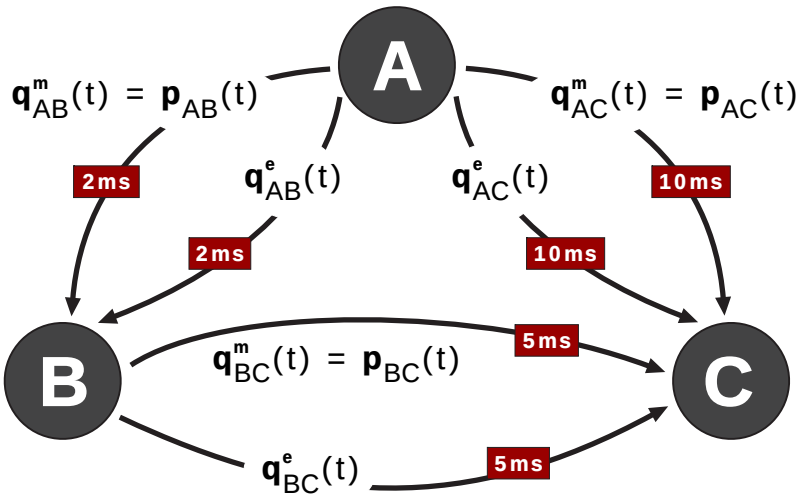



**Figure 2.3:** Spatial Relationship Graph according to relation $\Psi$, containing measured relationships as well as inferred relationships

In figure 2.3, the example Spatial Relationship Graph is shown, according to relation $\Psi$. The originally measured relationships are now represented by the function $q^m$, which is identical to the function $p$ in relation $\Phi$. Based on these measured relationships new relationships are inferred. Additional edges are inserted, annotated with function $q^e$, which represent the geometric information derived from extrapolation on basis of the measured relationships. The quality of the estimated spatial relationships is indicated by a single attribute, the latency of the measurement.

Assuming an application requests positional information about object $A$ relative to object $C$ at time $t = T$, this request can be fulfilled with the estimated spatial relationship $q^e_{AC}(T)$. Alternatively a new edge could be added to the Spatial Relationship Graph, annotated with function $q^f_{AC}(T)$, which represents the fusion of the estimations $q^e_{AB}(T)$ and $q^e_{BC}(T)$.

In order to choose an optimal estimation for the requested spatial relationship, the application defines an evaluation function which maps attributes on a real non-negative value. The spatial relationship with the best result value is returned to the application as result.

### 2.1.2 Attributes and Evaluation Function

Each edge in the Spatial Relationship Graph provides, in addition to the actual measurement values, attributes that describe the quality of the measurement. These attributes most importantly serve as selection criteria when multiple paths fulfill an application's request, but they can also be used as a weight when multiple measurements are averaged. Moreover, the user interface of an AR application can be adapted to the currently available tracking resolution, as demonstrated in [30]. Examples of such quality attributes relevant for augmented reality are:

**Latency** describes the time (in seconds) between the actual measurement and the availability of its result to the rest of the system.

**Update frequency** measured in $1/\mathrm{s}$, indicates the rate at which trackers make their measurements.

**Confidence value** attributes are important for optical trackers that may misclassify video images and detect non-existent features. A well suited range would be $[0; 1]$, indicating the probability that the identification is correct.

**Pose accuracy** determination is the major problem when developing a tracker abstraction to integrate multiple trackers [14]. A simple approach is the use of Gaussian noise models. The pose accuracy is then described by a covariance matrix which defines an (hyper-)ellipsoid around the measured position in which the real position lies with a probability of $68\%$.

**Monetary cost** per measurement may be significant in the design of tracking setups. The tracking graph can incorporate all the trackers available on the market and then a suitable evaluation function can be evaluated for the desired relationships between objects. The resulting optimal path consists of the trackers that should be installed.

**Motion models** are relevant when prediction has to be performed in order to compensate sensor and rendering delays in augmented reality applications or when short sensor dropouts have to be bridged. A motion model informs the Ubitrack system which derivations of the measurements to use for prediction, i.e. if the expected motion is constant positional property, constant velocity or constant acceleration. The motion model also describes how much the motion may diverge from this simple model.

A special case of a motion model is one which tells the system that the measured relationship between the two objects is static. In this case, the Ubitrack components will assume that all measurements are infinitely valid and not perform any extrapolation but use the value of the last measurement or an average over all measurements instead.

Although these attributes can be defined easily, it is often not trivial to obtain reasonable values from specific trackers. Manufacturers of tracking hardware usually only provide average or even best-case accuracy values in the documentation but do not make such information available in real-time for each measurement.

Another difficulty arises when measurements are transformed by inference or filter components, because the quality attributes of the resulting edge have to be adjusted accordingly. In case of confidence or latency this is easily to achieve, but more complicated attributes

such as pose accuracy require non-trivial methods of transformation. An approach for this is described in [48].

#### Evaluation Functions

In some cases, an application's query may result in multiple paths in the Spatial Relationship Graph that describe relationship between two objects. To resolve this, the developer of a Ubitrack application has to specify an evaluation function which is used to discriminate different solutions. An evaluation function takes a whole path, evaluates the attributes along its edges and computes a scalar value. This value expresses how well the solution suits the application's requirements. The optimal solution is associated to the lowest value. Equation 2.5 gives an example of an evaluation function that aims to trade-off latency against update rate.

$$e^t := \sum_{q \in path} \text{lag}(q) + \frac{\lambda}{\text{rate}(q)} \tag{2.5}$$

$\lambda$ determines the weighting of lag versus update rate. This example also shows that it is possible to create evaluation functions that computes a value from the attributes of a single edge and then sums them up along the path. In this case efficient search algorithms on weighted graphs, such as the algorithms of Dijkstra and Bellman-Ford [35] can be applied in order to find minimum-weight paths within the Spatial Relationship Graph.

### 2.1.3 Data Flow Graphs

The application's query for a spatial relationship results into a distinct path within the Spatial Relationship Graph. This result path determines a specific processing sequence according to which measurements have to be combined in order to infere the requested spatial relationship. This processing sequence is only dependent on the infrequently changing topology of the Spatial Relationship Graph and the quality of the measurements. Therefore, a data flow graph can be precomputed and updated in lengthy intervals which executes the real-time computation of the positional data.

This data flow graph is a tree of software components. Raw measurements are inserted at the leaves by tracking services and propagated to the root, undergoing various transformation at each node. The root is formed by the application which receives the computed measurements. The interior of the data flow graph consists of interpolators, extrapolators, filter, inference and data fusion components. This concept of a data flow graph can easily be mapped onto a network of DWARF services or OpenTracker nodes. An example of a data flow graph is shown in figure 2.4, the graph for the example scenario from section 2.3 is shown in figure 2.4.

A typical Ubitrack data flow graph consists of the following components:

**Tracking components** provide a stream of measurements, consisting of positional information and the quality of the measurement. The measurements are usually taken by tracking device such as optical trackers or inertial trackers. Alternatively, previously recorded or simulated positional data can be provided for algorithmic evaluation.

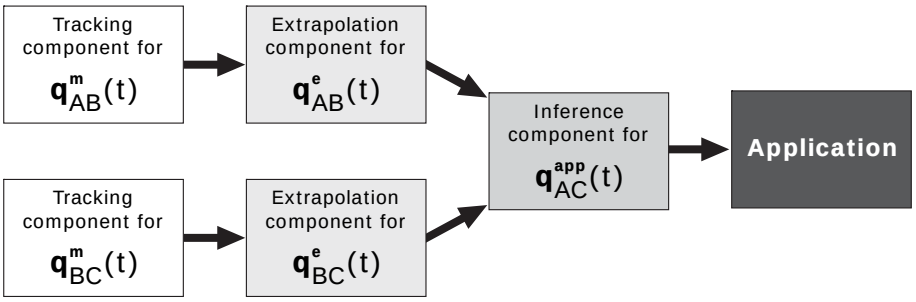



**Figure 2.4:** Example of a data flow graph

**Static components** provide static measurements, e.g. from a database, consisting of positional information and the quality of the measurement. This information can be derived from previous measurements, e.g. by averaging. But static components can also be configured manually. Examples of static measurements occurring frequently in AR systems are the relationships between fixed markers on a wall or the offset between an HMD and an attached camera for optical tracking.

**Extrapolation, interpolation and filtering components** take a stream of measurements and perform the according transformations on the basis of the initial measurements. The resulting positional information and the adapted quality of the information is reintroduced into the data flow graph.

Extrapolation and interpolation are of particular importance in a Ubitrack system. New spatial relationships can only be inferred by combining measurements along a path which have been taken at a distinct same time. But in inherently heterogeneous Ubitrack systems it is rarely achieved to provide measurements of different spatial relationships at the same point in time. Therefore interpolation or extrapolation must be applied in order to compute measurements from the raw sensor data, that expresses the different spatial relationships a the requested point in time. A well-established tool for extrapolation is the Kalman filter which smoothes noisy data as a side effect [4], [61].

**Inference components** are the heart of a Ubitrackdata flow graph. They take two or more measurements that correspond to the edges along a path in the spatial relationship graph and compute the inferred relation from the beginning of the path to it's end. As all of the input measurements must be valid at the same moment, inference components in the data flow graph are usually preceded by inter-/extrapolation components.

**Fusion components** take input measurements from at least two different sensors measuring the same relation and compute a new combined measurement that ideally should be better than each of the inputs alone. In the simplest case the measurements are weighted by their error covariance. More complex sensor fusion setups can combine sensors with different characteristics. For example, an optical tracker that provides high precision measurements, but suffers from high latency and casual dropouts can be combined with an inertial tracker that drifts, but has low latency at a high update rate [4], [64].

**Pose inversion components** take a measurement as an input and compute the inverse relation. When an application requests the relation between two targets $A$ and $B$ that are tracked by a single optical tracker $T$, providing the measurements $A \rightarrow T$ and $B \rightarrow T$, $B \rightarrow T$ has to be inverted before the requested relation $A \rightarrow B$ can be computed by an inference component.

## 2.2 Dwarf

The Distributed Wearable Augmented Reality Framework (DWARF) is a component-based framework which allows the rapid prototyping of AR applications. It was developed at the Chair of Applied Software Engineering of the Technische Universität München (TUM) [6, 19]. DWARF provides usage of distributed components on several machines and ad-hoc connection of these components by a middleware. This leads to a higher flexibility in the development of new AR applications. In combination of aspects of Ubiquitous Computing it is possible to build a new AR system using already existing components: e.g. a component for tracking or for displaying synthetic visual data can be used not only by one application but by multiple applications. Depending on the properties of the components a simultaneous usage can be obtained.

This section gives a short overview of the concepts of DWARF restricted to those which are relevant for Ubitrack. It focuses on introducing the high-level modeling concepts and communication mechanisms.

### 2.2.1 Services

Applications implemented with DWARF use interdependent, collaborative *services*. These services are software components which may be distributed over the system's underlying network infrastructure. Every service runs as a discrete unit, mainly as one process, on a network node. The whole application is formed by combination of multiple services which are performing certain tasks.

Every service is determined by its *service description*. These descriptions are stored in XML files which are evaluated by a middleware component, the DWARF service manager (see section 2.2.3). They contain all relevant information about the service, e.g. the provided features or required input data which must be provided by other services. The provided features are modeled by *abilities* whilst required input data are modeled as *needs*.

### 2.2.2 Needs and Abilities

Abilities are features which are offered to other services. The service's demand for such a feature is modeled as need. According to this, needs and abilities can be connected in a way that the provided features of one service meets the demand of another service. The specification of service's needs and abilities is defined as in the following:

1. The *name* is an identifier, unique within the according service
2. The *type* describes what kind of information is delivered or requested. If and only if the type of a need and a ability matches, the communication between them is established.
3. The *connector protocol* describes the requested communication mechanism.

### Attributes

Services can specify certain *attributes*[1] in the descriptions of their needs and abilities. Attributes are generic name-value pairs which can contain arbitrary information. Usually attributes are used to give a high-level description of provided features or details to the quality of the available data. The attributes can be evaluated by the service itself to adapt its behavior at runtime. But mostly they are used by needs to limit the number of matching abilities. Therefore needs specify a certain predicate to evaluate the according attributes of possible partners.

### Need Predicates

*Predicates* restrict the number of possible partners for a need. The attributes of a candidate partner are evaluated according to a Boolean expression specified in the predicate. If and only if the partner service's attributes meet the demand of the need's predicate, the according service's ability is connected to the need.

### Connectors

Every need or ability has one or more *connectors*. For every communication method exist a connector protocol that must be named in the description. DWARF service can currently use four different communication methods.

**Remote Procedure Call** Services can use CORBA for a inter-process and inter-network communication. CORBA, the Common Object Request Broker Architecture, is a platform-independent, object-oriented architecture and infrastructure standardized by the Object Management Group (OMG) [45]. By using the Interface Definition Language (IDL) the developer can define remote class interfaces which are called by remote method invocations. The connector protocol for this type of communication is called `ObjrefExporter`, because this services provides objects which can be imported on the remote side. Thus the partner service uses the `ObjrefImporter` connector protocol.

**Event based Communication** Additional to remote procedure calls, CORBA provides a notification service to enable event-based communication between services. Because of the asynchrony of event-based communication, this is the most common form of communication between services. Messages are packed into CORBA structures and delivered over an *event channel* to the communication partner. Within these CORBAstructures arbitrary data can be sent as long as the communication partner can process the structure correctly. Therefore needs and abilities specify in their *type* value the data type stored in the CORBA events which are used for communication. The connector protocol for sending events is defined as `PushSupplier` whereas receiving services use the `PushConsumer` connector protocol.

[1] DWARF attributes are distinct from the attributes describing the quality of measurements within a Spatial Relationship Graph 2.1.2. However, DWARF attributes can be used to propagate the quality of measurements within the system. In this case the quality attributes are represented by DWARF attributes.

**Shared Memory** If communicating services are running on the same computer, communication between them can be established by using shared memory. One service can write the desired information in a POSIX shared memory block where the other service has access to and can extract the information from. DWARF only provides mechanisms for allocation of the memory area and for synchronization; the concurrent access to the memory blocks must be handled by the services themselves. In contrast to the above mentioned communication methods only one connector protocol, the `Shmem` protocol, is used on both endpoint of the connection.

**Communication-less Connection** In some cases a connection between needs and abilities is desired without to provide a communication method. For this reason the `Null` connector protocol is defined.

### Data Flow between Dwarf Services

Generally a service providing data establishes an according ability; the requirement for data is therefore defined by a service's need. But in some cases it is reasonable to reverse the direction of the data flow, so that needs are providing data and abilities are receiving data. Therefore the direction of the dataflow is not coupled with the need-ability-relationship, but only with the definition of the connector protocol in the needs and abilities. This is supported, since the connectors can be independently set for a need or ability.

### IDL interfaces

The connectors describe the communication method, but the actual specification of the used interfaces is accomplished in CORBA's Interface Definition Language (IDL). Each of the three available connector protocols for inter-service communication use one or more IDL interfaces to implement the communication method. IDL is an type description language which allows to specify object-oriented class interfaces. Mappings of IDL exist for several programming languages including C++, Java and Python.

Any component using a specific protocol may implement an arbitrary IDL interface. However the remote end must know which interface may be called. This information is stored in the ability type. Every interface is designed for a specific ability. So the remote end knows what interface can be called since it has the same ability type.

### UML Notation

Services and need-ability-relationships can be modeled by UML diagrams used for representation of the system design of DWARF applications. The special DWARF notation extends the common UML by defining a new need-ability-relationship association. UML diagrams are used throughout this thesis for representation of the DWARF models.

Abilities are drawn as filled circles attached by a line (similar to UML interfaces), needs on the other hand are semi-circles. Both are attached by a UML dependency arrow, where the need always depends from the matching ability.

Figure 2.5 shows three example services. Only two of them are connected since the ability attributes of one service do not match the given predicate of the need of the other.

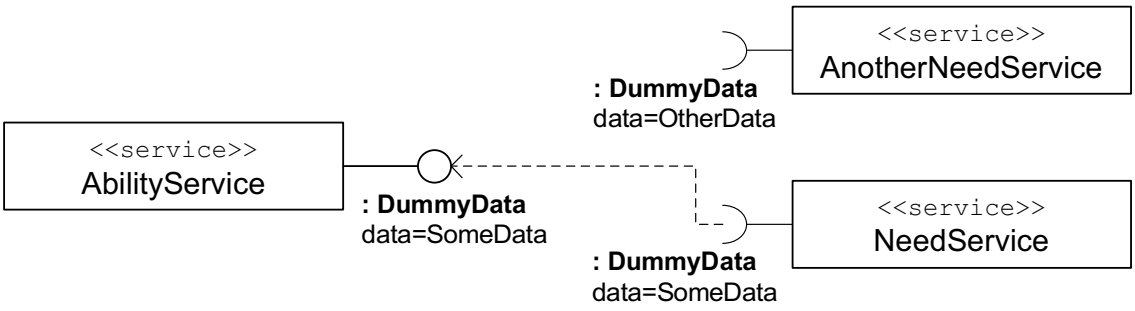



**Figure 2.5:** This UML diagram shows an example session of three services. Two of them are connected, one is not since the predicate does not match. The annotations in the figure describe the individual UML notations.

### 2.2.3 Middleware

The DWARF *service manager* allows the development of distributed DWARF applications. At each node of the network one service manager is located and dynamically finds all other service managers in the network. Each service manager aggregates information about locally available services by parsing the local XML service descriptions. When a local service is started, it *registers* itself at the according service manager and changes into an active status. If it the service defines a need, the service manager tries to find a service with a matching ability by requesting other service managers for potential partners. If a matching partner service is found both service managers establish the desired connectors and create a direct communication channel between the services.

Services can specify a start-on-demand flag. Is this flag set, the service manager can start the service by itself, if another service requests for an ability of this service. This feature can be used to start whole *service chains* by manually starting only one service. Initiated by the manually started service, the service manager starts recursively all other services of the chain by itself.

### 2.2.4 Dwarf Template Services

This mechanism is only working when using DWARF *template services*. The Service Manager can maintain Service Descriptions where attributes may contain wildcards. They act as placeholders for arbitrary values. This is sensible when abilities can adapt to certain predicates of needs. The wildcard in the Service Descriptions is indicated by the asterisk "`*`".

For example, a tracker service has the ability `PoseData` with the attribute `updaterate=*`. The value of this attribute is not set until a corresponding need exists. If a service needs a tracker with specific update rate of 60Hz, it has a need with the predicate `updaterate=60Hz` The Service Manager checks the ability and notices that the attribute is wild-carded and that it is a template service. Therefore, it instantiates the service with a specific Service Description where the attribute is set to 60Hz. Because the predicate of the need service and the attribute of this new *clone service* match, they are connected (figure 2.6[2]).

[2]For simplicity this distinction is not often shown in DWARF UML diagrams. If a service contains wildcard attributes, you can assume that the template service is meant, Otherwise it is a normal or cloned service.

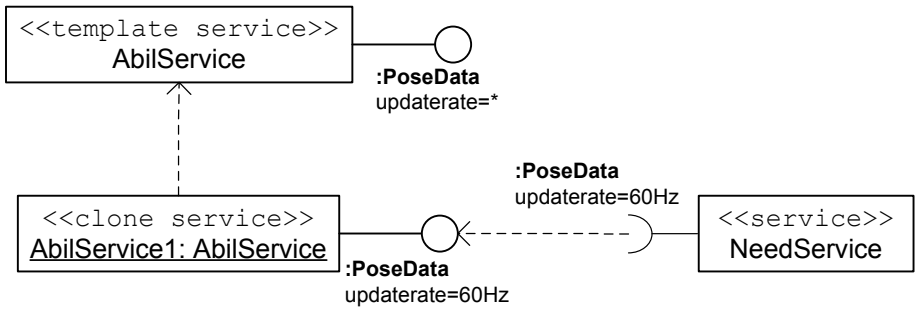



**Figure 2.6:** The UML diagram of a template service and its clone

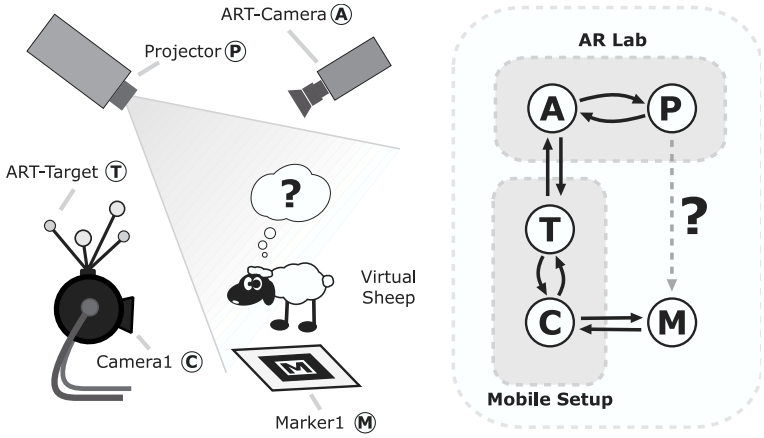



**Figure 2.7:** The example setup and its spatial relationship graph

## 2.3 An Example

This section introduces a simple example scenario which illustrates the Ubitrack concepts. This example is used to explain how the Ubitrack concepts are mapped onto DWARF in the rest of this chapter. A stationary tracker combined with a mobile camera delivering images to an optical tracker effectively allows the system to "see around corners". Figure 2.8 shows a camera tracked by an ART tracking system. Figure 2.7 shows both a schematic of the hardware setup, and the corresponding Spatial Relationship Graph. A ceiling-mounted projector $P$ displays a pastoral landscape, populated by a sheep, on a table. The projector is calibrated so that it shares the same coordinate system as the ART tracker $A$; therefore, a static relationship exists between $P$ and $A$. A mobile user is equipped with an ARToolKit [32] camera $C$ on which an ART target $T$ is mounted, resulting in another static relationship described by another edge in the graph. The camera is attached to a notebook with a wireless network interface. The middleware is running on the lab computers and on the notebook.

When a user enters the room, the two Ubitrack systems connect and the description of an ARToolKit marker is transferred to the user's notebook enabling the camera to track it. As long as the marker remains in the view of the camera, and the ART target attached to the camera is tracked by the ART system, the virtual sheep can still be tracked in the coordinate frame of the ART system, even if it is physically out of range. Consequently, when the marker is moved, the image of the sheep displayed by the projector can be seen to move

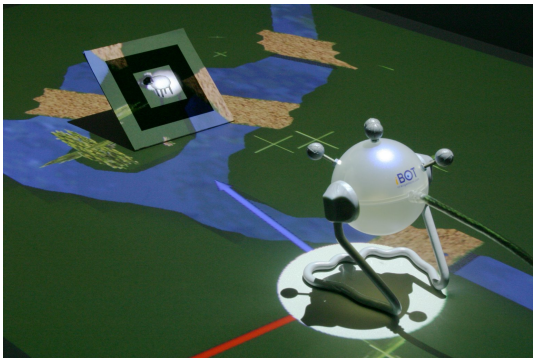


**Figure 2.8:** Picture of an example scenario implemented in DWARF: An AR Toolkit marker is tracked by a mobile camera which is attached to an ART target. A virtual sheep is projected onto the AR Toolkit marker relative to the ART coordinate system.

accordingly. If the sheep is moved into a pen outside the range of the projector, it can still be viewed using some other tracked or fixed display.

The combination of these two tracking technologies is not novel in and of itself; however, this example focuses on the inference of spatial relationships and the resultant spontaneous behavior of the system as it reacts when tracking sensors are combined.

## 2.4 Modeling Ubitrack in Dwarf

Based on the example scenario a suitable model for DWARF was developed. This model takes the Ubitrack concepts and maps them onto existing DWARF techniques, resulting in the integration of existing DWARF components with new ones. This section only introduces the new key concepts. They are explained in the different diploma theses of the Ubitrack project in more detail.

### 2.4.1 Sensor API

The Sensor API is the interface between the hardware and the Ubitrack layer, which all services that control a sensor and deliver positional information have to implement. The interface is mapped to an ability with type `PoseData`. Every measurement, i.e. every edge in the Spatial Relationship Graph is mapped onto exactly one ability of such tracking services. So every ability has a unique pair of source and target *object identifiers*, the corresponding nodes of the edge. In DWARF, these identifiers are stored in attributes called `UTSource` and `UTTarget`.

This ability delivers positional information in form of an IDL data structure of type *PoseData*. This IDL structure contains the measurement values from the sensor hardware, i.e. spatial measurements of objects in the frame of reference of the sensor. Table 2.1 gives a review of the data fields which are defined by the current Sensor API. However depending on the type of sensor, not all fields contain sensible data.

| Field | Description |
|---|---|
| Position | The 3D position of the object is stored in an array of double floats in Cartesian coordinates. |
| Orientation | The orientation of an object is delivered in form of *quaternions*. Quaternions are a non-commutative extension of the complex numbers. They may be represented by 4-dimensional vector $q = w + xi + yj + zk$ where $w$, $x$, $y$ and $z \in \mathbb{R}$. $i$, $j$ and $k$ are the imaginary units with $i^2 = j^2 = k^2 = ijk = -1$. Rotation can be expressed with them as shown in [20]. $x$, $y$, $z$ and $w$ are stored in that order in an array of double floats. |
| Source | This string contains the sensor's object identifier in the Spatial Relationship Graph. |
| Target | This string contains the target's object identifier. |
| Timestamp | A data structure containing the absolute timestamp of the measurement expressed in the time frame of the measuring system. |
| Confidence | A statistical value giving the probability that this measurement exists. |
| PoseError | A 6x6 covariance matrix parameterizing the Gaussian error distribution of the measured position and orientation. |
| Flag fields | There are some flags which indicate what fields of this structure are set with valid data. |

**Table 2.1:** Data fields which are stored in a PoseData structure

### 2.4.2 Query API

Applications receive spatial information from the Ubitrack layer through the Query API. In DWARF this is realized by a need of type `PoseData`. What information is desired depends on the predicate of the need. Required predicates are `UTSource` and `UTTarget`. Given a certain evaluation function, quality attributes can also be specified in the predicate which restricts to tracking services with the given accuracy, for example.

### 2.4.3 Query mechanisms

The connector of the need implies the delivery mechanism for the results. There are three different query mechanism which work synchronously or asynchronously, with the push or pull principle.

#### Asynchronous Push

This is the most common query mechanism in current DWARF application for PoseData. It uses the CORBA notification service for sending PoseData events whenever a measurement is possible. Depending on the sensor technology this can take place in a constant update rate or exactly on recognition of a locatable. The `PoseData` ability uses a `PushSupplier` connector and probably sets an attribute containing the update rate in Hz ($1/s$).

Every ability must implement the `UTPoseDataAsyncPush` IDL interface, which is a derived interface of the `SvcProtPushSupplier` interface. Latter is used in current DWARF

implementations as default interface for `PushSupplier` connectors. The use of a derived interface in Ubitrack allows future extensions to the asynchronous push mechanism.

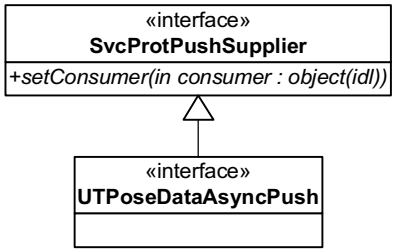



**Figure 2.9:** The `UTPoseDataAsyncPush` interface

### Synchronous Pull

The synchronous pull is modeled by a remote method call, so it uses the `ObjrefExporter` connector to offer its interface for the application. The IDL interface which must be implemented for this pull mechanism is the `UTPoseDataSyncPull` interface. This interface currently consists of only one method: `getPoseData`. The method gets a timestamp as parameter and blocks the execution until a result is available.


**Figure 2.10:** The `UTPoseDataSyncPull` interface

### Asynchronous Pull

The most complex interface is the asynchronous pull interface: `UTPoseDataAsyncPull`. It aggregates the `SvcProtPushSupplier` interfaces as well as its own method: `wantPoseData`. This method takes a timestamp and returns immediately after a PoseData event for the given time has been scheduled. When a PoseData event is available it is delivered via the event channel.

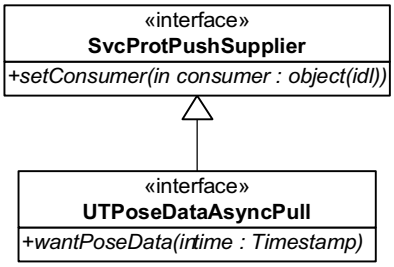



**Figure 2.11:** The `UTPoseDataAsyncPull` interface

### 2.4.4 Ubitrack Middleware Agent

The *UbitrackMiddleware Agent* (UMA) is a distributed middleware component. Allocated in the Ubitrack Layer of the architecture model, it focuses on three aspects of the Ubitrack concept:

- Aggregation of information about locatable objects and their geometric relationship to build a distributed representation of the Spatial Relationship Graph.
- Aggregation of inferred spatial information between objects requested by applications by executing a distributed path search in the Spatial Relationship Graph.
- Set up of data flow to provide the requested spatial information to an application.

#### Representation of the Spatial Relationship Graph

On each host in the network, a single UMA maintains a representation of the local Spatial Relationship Graph. Edges in this graph are associated with the abilities of software services to provide measurements of the according spatial relationships. These services have a `PoseData` ability implementing the Sensor API. Attributes describing the quality of the estimated positional information are mapped on the ability attributes. In order to reflect the infrequently changing quality of the estimations, these attributes have to be adjusted in lengthy interval.

In order to obtain information about spatial relationships, the UMA has to parse the service descriptions of local services providing measurements of spatial relationships. Therefore the UMA gets connected to the local service manager, which obtains all service descriptions of locally running services. For initialization, the UMA polls the list of available service descriptions, but after this the UMA receives notification messages from the service manager whenever the properties of a locally running services change. The notification messages also indicate the start up of new services and the shut down of already running services.

An first inference process can extend the knowledge of the spatial relationships by inverting relations and adjusting the corresponding attributes. This leads to a set of inverse edges and results in an undirected Spatial Relationship Graph.

Locatable objects can take part in multiple spatial relationships stored in different UMAs in the network at the same time. UMAs storing spatial relationships of identical objects need to interact in order to find optimal estimations of requested spatial relationships by performing a distributed path search. A bidirectional event channel must be established between them for passing messages during the search process. The UMA indicates for which objects it requires connections to send and receive search messages by configuration of the according needs and abilities in its own service description. The connection of matching needs and abilities, and thereby neighboring UMAs, is done by the DWARF middleware.

This distributed representation of the Spatial Relationship Graph contributes to the systems scalability and flexibility. Centralized storage of the Spatial Relationship Graph or providing coherent copies at each host would lead to an immense communication overhead and long response time to changes in the topology. In a distributed representation, the AR system is easily to extend and mobile clients equipped with their own tracking setup and running AR application can be integrated seamlessly.

#### Distributed Queries and Path Search

An application indicates a request for a specific spatial relationship by implementing the Query API in a service need. By parsing the service description of the application, the local UMA obtains information about the properties of this request.

The path search between the UMAs is implemented using a distributed asynchronous Bellman-Ford algorithm [35]. In a first step, the path search is performed on the local subgraph. Even though a suitable estimation of the requested spatial relationship might be found in the local subgraph, there is no certainty that this estimation is the optimal result to the application's query. Therefore the path search must be extended to all adjacent UMAs.

Search requests are sent out containing following properties: a search identifier, the current node visited during the path search (initially this is the source node), the target node, and an evaluation function that computes edge weights from attribute sets. By evaluating the attribute sets along the path from the source node to another node, a distance value is obtained on which Bellman-Ford is based. This distance and the according path are stored additionally. When a node is visited during a path search, the distance of this node from the source node is updated. When a search computes a new minimum distance between a node and a source node, this new distance must be propagated to neighboring UMAs. Therefore a participation identifier is stored additionally for each distance update that is sent out.

When a search event is received by the UMA, the distance associated with the current path is compared with those from earlier participations. If the new distance is less than that of former participations, the path search must be continued from this node. The distance from the current node to all other local nodes is updated by computing minimum-cost paths and new search events are sent out to relevant UMAs.

If the distance is greater than that of the current node or the target node is found, an acknowledgment message is sent back. Each time a UMA participates in a path search, it must wait for acknowledgment messages or a timeout before sending an acknowledgment message back on the path towards the source node. After all search messages, sent out by the UMAs containing the source node, have been acknowledged, the UMA chooses the over-all minimum-cost path.

#### Set Up of Data Flow

After the path search results in an appropriate path, the UMA sets up new Data Flow Components by configuring their service description according to the result path. The Data Flow Components are started by the local service manager and connected to running tracking services. They form the inner part of a data flow graph, combining measurements provided by tracking services and delivering the inferred spatial data to the application.

### 2.4.5 Data Flow Components

In the DWARF Ubitrack implementation, all nodes of the data flow graph are realized by distinct DWARF services. The root of this graph consists of the application, represented by a service with a need for `PoseData`.

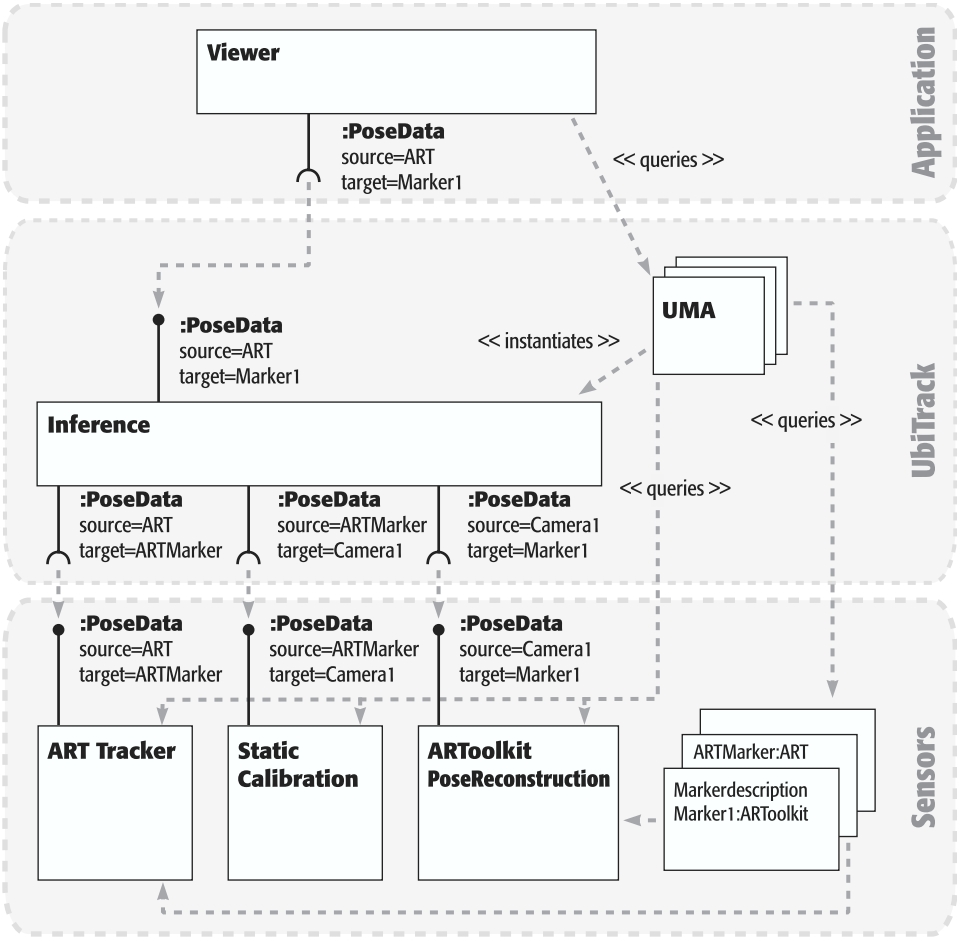



**Figure 2.12:** The example scenario modeled by DWARF services (which are data flow components)

### Leaf Components

At the leaves of the data flow graph are the tracking services which provide raw sensor measurements. Tracking services usually send out a PoseData event as soon as a new measurement is made using the asynchronous push protocol. When a component at a higher level in the data flow graph requires measurements at arbitrary times, a Kalman filter component has to be inserted for interpolation or extrapolation. The leaf components are of the following categories:

**Trackers** make real measurements of the relation between two objects. Available tracking services include ART DTrack, Intersense and AR Toolkit. All of them were already available, but we adapted them to send measurements in the new extended Ubitrack *PoseData* format.

**StaticCalibration** sends out a static measurement at a fixed interval. The value of the measurement is determined by the `Position`, `Orientation`, `Covariance` and `Confidence` attributes. The `UTUpdateRate` attribute specifies the interval at which the measurement is sent out. In addition to the asynchronous push protocol, the `StaticCalibration` service also supports synchronous pull, as the static measurement is considered valid at all times.

**PoseDataPlayer** service inserts previously recorded measurements into the data flow graph. This is useful for offline data analysis or for algorithm evaluation where repeatability is extremely important in order to obtain comparable results. Sequences of measurements can be recorded using the `PoseDataLogger` service.

### Interior Components

The interior components in the data flow graph perform all the measurement transformations that are necessary in order to deliver inferred, fused and filtered measurements to the application. Each data flow graph component is realized by a single DWARF service and performs only a small part of the whole computation to allow reuse in more complicated graphs.

The service descriptions are created by the UMA in response to an application query or a change in the sensor topology. Service descriptions act as a configuration for the services and make sure that the services connect to the respective partners in order to form a given data flow graph. Interior services communicate with each other mainly using the `UTPoseDataSyncPull` protocol. All interior data flow graph services run in a single process space for higher system performance.

The following list gives a short overview over the implemented services. A more detailed description including the underlying mathematics can be found in Daniel's thesis [48].

**Kalman Filter** services mainly provide a contiguous-time `UTPoseDataSyncPull` protocol to those leaf components that only make measurements at discrete time intervals by sending events over the `UTPoseDataAsyncPush` protocol. The Kalman Filter maintains an internal state of the relation, usually including position, orientation and a number of derivations. This state is used to compute measurements at past or future moments as they are demanded by other components in the data flow graph. The

filter also maintains a queue of old measurements which are interpolated when a measurement is requested at a time earlier that what can reasonably be derived from the current internal state.

A Kalman Filter service instance can handle an arbitrary number of input needs that all measure the same relation. In this case data fusion is performed by integrating measurements from all sensors into the internal state. This way, trackers with different characteristics can also be combined, e.g. inertial and absolute sensors.

Usually the Kalman Filter receives new measurements using the `UTPoseDataAsyncPush` protocol and each incoming measurement is immediately integrated into the internal state. For cases where the filter is used for sensor fusion inside a more complex data flow graph, the `UTPoseDataSyncPull` protocol is supported, too. The filter output can only be requested via `UTPoseDataSyncPull`.

When the internal Kalman Filter state is updated by a new measurement, some assumptions about the type of motion are required to be able to decide how to weight the measurement with respect to the internal state. If more weight is given to the internal state, a bigger portion of the incoming measurements is considered noise, resulting in a tighter filtering. This leads to the specification of a motion model, which is included with every relation in an additional DWARF attribute `UTMotionModel`. When no motion model is specified, a default model using first and second derivations of both position and orientation is used.

**Inference** services are responsible for combining multiple measurements along a path, which is one of the core principles of the Ubitrack framework. An inference service instance may have an arbitrary number of input `PoseData` needs and has one ability for providing the resulting computation. As all input measurements must be made simultaneously, the inference only supports the `UTPoseDataSyncPull` protocol for both input and output.

**MeasurementInverter** services invert measurements when edges in a path point in the wrong direction. They have exactly one input and one output. Two different implementations exist for the `UTPoseDataSyncPull` and `UTPoseDataAsyncPush` protocols.

**MeasurementSampler** services are inserted at the end of a data flow graph if an application prefers to be notified of new measurements at a fixed update rate via the `UTPoseDataAsyncPush` protocol. The service has a `UTPoseDataSyncPull` need, usually connected to an inference or Kalman filter, and an `UTPoseDataAsyncPush` ability. The update rate at which events are generated is specified by the `UTUpdateRate` attribute at the receiver's side.

A MeasurementSampler also generates the correct timestamps for prediction. This behavior can by controlled by the receiver using the `UTTimeOffset` attribute.

## 2.5 Related Work

VRPN [55] implements an abstraction layer for a wide range of virtual reality devices, including trackers. It provides a network-transparent client-service interface with time stamps and clock synchronization between multiple computers. Supported device types include

position and orientation sensors, buttons, dials, analog and force feedback devices. Layered devices can be implemented in order to process data from multiple sensors. VRPN however has no notion of different coordinate systems and transforms nor does it identify multiple trackers and tracked objects.

OpenTracker [49] implements a "pipes & filters" data flow model for the processing of tracker measurements. Source and sink objects set up connections to trackers and applications and allow the transport of sensor data over the network. Filter nodes perform static and dynamic coordinate system transformations or smooth measurements in order to remove noise.

Many previous research projects deal with data fusion of different sensors in order to improve tracking stability and accuracy in static sensor setups [4, 22, 65]. Hoff [29] also describes a setup that combines measurements from fixed and user-worn optical trackers. However, no attempt has been made to dynamically integrate arbitrary sensors.

The so far the comparable results come from Höllerer, Hallaway et al. [25, 30]. They describe a system that combines a high-resolution IS 600 tracker with wide-area infrared beacon observations. When none of these trackers is available, the gaps are bridged by a dead-reckoning technique using inertial orientation sensors, a pedometer and environmental knowledge. A Kalman filter is used to fuse measurements from all tracking systems. However, the system is limited to this particular tracking configuration and a single user as they do not have a software architecture to dynamically integrate arbitrary sensors. All sensors must be calibrated to use the same coordinate system. The group also describes an interesting approach for adapting the user interface of a personal navigation application to the currently available tracking resolution.

# 3 Bootstrapping of Tracking Systems

**Why is bootstrapping in large-scale sensor networks reasonable? A concept of bootstrapping Ubitrack networks is introduced and the focus of my work is defined.**

---

As mentioned in the introduction, large sensor networks as proposed in the Ubiquitous Tracking chapter are hard to manage. Many devices are mounted in the environment and controlled by dedicated software. These devices are invisible for the users, so they cannot control them directly. Furthermore, one cannot clearly say what device is currently working for what applications because they can be used by more people at the same time. Therefore manual configuration of Ubitrack services must be avoided whenever possible.

## 3.1 Motivation

Trackers in classical AR systems are deployed over the whole area of interest where information should be gathered. If they are deployed statically and their infrastructure does not change, the application which uses the environment can be pre-configured. These applications are bound to the specific tracking environment almost certainly. This is however not the goal of Ubiquitous Tracking: it should provide a large-scale and flexible sensor network. Therefore following characteristics for a large-scale Ubitrack environment have to be considered:

**Exchangeable Components** Sensors can be replaced on the fly when the application is running. For a robust application, sensors should be exchangeable like light bulbs. Probably the controlling software has also to be replaced. Therefore we need a component-based system where software parts can be exchanged as easily as hardware.

**Scalable for Large and Complex System** The assumption is that the system is distributed over a network of node computers. Every computer runs the software components for the connected sensors. The components should start automatically because the resulting overall system may consist of hundreds of them. This is realized by *bootstrapping* mechanisms. This complex system has to be divided into smaller parts which are easier to manage. By grouping sensors by their spatial neighborhood, a *location context* for nodes in the Spatial Relationship Graph is built. Since sensors may be mobile, changes of the context has to be propagated to the mobile set-up.

**Self-Describing and Configurable Components** The components have to be able to describe themselves. Since they are relatively independent units they have to be able to be

configured by suitable configuration data, provided by others. So the overall system is configured by itself.

**Pseudo-Realtime** Changes in the infrastructure should be propagated as fast as possible. The desired goal is realtime reconfiguration of all components. This is due to the fact that delays in the reconfiguration of the system inhibits the correct execution of applications since they rely on working sensors.

**Mobile Set-ups** A flexible infrastructure can most obviously be realized when it supports the integration of mobile tracking set-ups at runtime in a running ambient system. Therefore I will distinguish between *mobile clients* and *stationary set-ups*. A mobile client is a simple tracking system which has to be integrated in a stationary system which then is extended to a larger and more complex one.

### Conclusion

A large-scale and complex Ubitrack environment must be set up by using simpler subsystems. The final components may be grouped in these subsystems depending on the location context. So the problem is to allow whole subsystems to build a complete system without even knowing each other. They cannot know each other initially because if they do, the complete system is already formed. These types of problems are known as bootstrapping problems.

## 3.2 The "Bootstrapping" Problem

The phrase *"bootstrapping"* is common overall in computer science fields, but it does not always mean the same. A general definition can be found at Wikipedia[1]:

> "In computers, this term refers to any process where a simple system activates a more complicated system. It is the problem of starting a certain system without the system already functioning. It seems just as impossible as "pulling oneself up by the bootstraps" which Baron Münchhausen, according to stories, could do.
>
> However solutions, accordingly called *bootstrapping*, exist; they are processes whereby a complex system emerges by starting simply and, bit by bit, developing more complex capabilities on top of the simpler ones."

This definition means that systems do not have the initial knowledge to start themselves completely. This can be solved by starting components stepwise which start others, until the whole system is formed. This problem is as old as computer science itself. In this section I will present examples of bootstrapping before a definition for Ubitrack is given.

[1] Wikipedia homepage: http://www.wikipedia.org

### 3.2.1 Classical Bootstrapping

As you can see, bootstrapping is a method to incrementally start a complex system. Since this is a very general definition, many methods exist which are called bootstrapping mechanisms. I present samples where bootstrapping is used.

I will list some examples starting from easy understandable PC bootstrapping to setting up complex networks. They will show the principals of my work step by step.

#### Bootstrapping a PC System

Booting a PC system is the most popular application of the described method. The general idea is to start the operating system of a PC by loading the kernel step by step. First of all, the BIOS, an integrated program on every PC, is started. It provides basic interrupt-driven routines to the hardware. The next step is to load a bootloader, a program residing on a floppy or hard disk. The bootloader locates the kernel and tries to start it. After the kernel is loaded, the operating systems starts [39, 46].

I have chosen this example to demonstrate the basics of bootstrapping. Everybody who is familiar with PCs should be familiar with this concept.

#### Ad-Hoc Networks

The invention and usage of wireless network technologies allow new possibilities of mobility. Since mobile users enter formerly unknown infrastructure, they cannot pre-configure their network clients. The clients themselves have to ask the environment how the environment looks like. This seems to be a paradox, but you can solve this by the assumption that there is a service which can be detected and which sends information about the network infrastructure.

The first step to form an ad-hoc network is to find the partners. This sounds simple and there are many different approaches for it. For example in wireless networks, we can distinguish between the *managed* mode and the *ad-hoc* mode. Whereas the managed mode relies on central access points, the latter builds networks by using direct peer-to-peer connections. These channels are the data link layer of the ISO OSI networking layer model.

After the partners are found and the communication channels are established, addresses for the network layer have to be assigned. This can be done by manual configuration or by detecting a component which assigns the addresses. The most common way to assign addresses is the *Dynamic Host Configuration Protocol* (DHCP) [18]. It manages a repository of IP addresses and assigns them to clients which request them in a broadcast. The only requirements the client has to fulfill are that it runs a DHCP software client and that the data link layer uses the Ethernet protocol. DHCP itself is part of a more generic protocol. *Zeroconf* or *Zero Configuration Networking* [66, 62] is an IETF standard protocol for dynamic configuration of network nodes in an IP network. Work on the standard is still in progress and no RFC documents have yet been released, though several implementations of the protocol already exist. Apple Computer implemented a first version of the protocol: Apple's *Rendezvous* framework [50].

The next step is forming the topology. Network nodes they must know their current neighbors or central gateways. In network terms this is called *routing*. Current researches try to

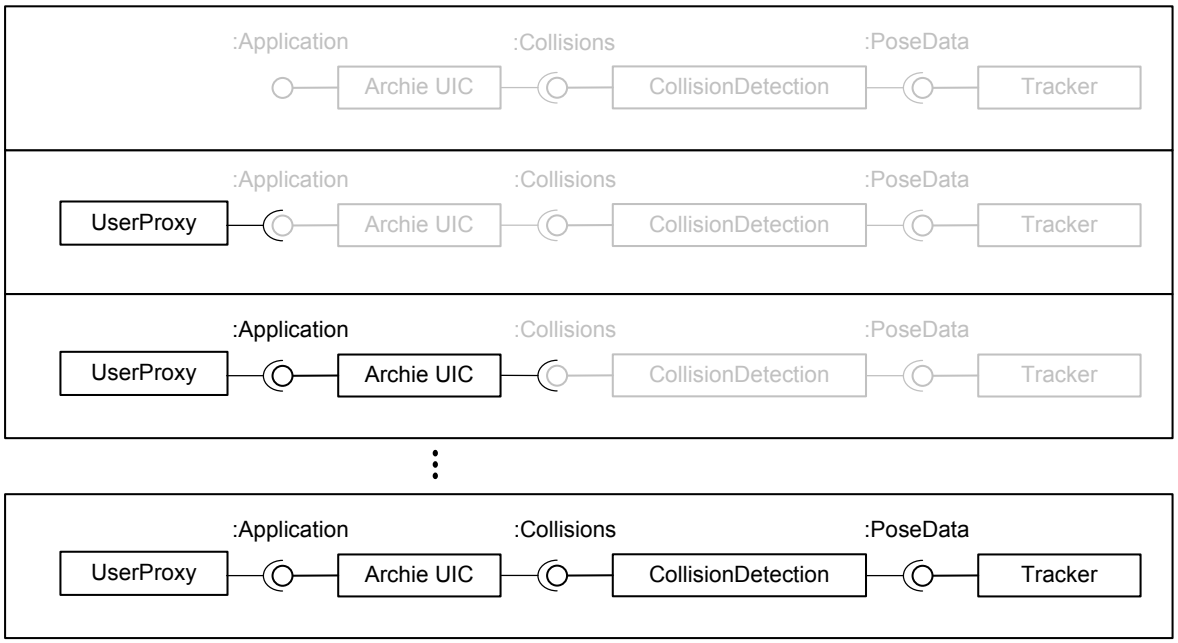



**Figure 3.1:** With the start of the `UserProxy` service a whole service chain is started on demand. This is a simple example of booting whole DWARF applications. Further information can be found in [53].

develop intelligent routing mechanism that are based on peer-to-peer (P2P) technologies, for example [47].

### 3.2.2 Bootstrapping Dwarf applications

Bootstrapping can also be used in DWARF applications. The main purpose is to start service chains on demand depending on their need-ability relationships.

The repositories of DWARF Service Description managed by the Service Manager s contain all information about the possible relationships between services. The Service Manager loads the repository when it is started. That way, it is aware of all possible relationships between services. The current implementation of the Service Manager offers for that purpose a useful feature: starting services *on demand*. If a need of a service can be satisfied by another service (since it provides a suitable ability), the Service Manager can automatically start this service. This can be repeated until no needs can automatically be satisfied or until the service chain is complete, i.e. there are no unsatisfied needs anymore. By introducing a service which has a high-level need for an application [53] whole applications consisting of complex service chains can be started on demand. So small and simple services can boot complex systems.

For example, in the ARCHIE demo [2], the `UserProxy` service has an `Application` need. It runs on a wearable computer of the user and acts as a placeholder for the user. The interaction component, the User Interface Controller (UIC), has the corresponding ability. When the users enters the room where the desired application is running, the `UserProxy` service starts the `UIC` service. The UIC then starts the services which it needs for itself. The schematic example can be seen in figure 3.1.

The possibility to start services on demand can be used in Ubitrack to implement compo-

nents which only exist to start other services. When they are detected by the DWARF Service Manager it starts the depending chain and stops it when the connection to the this service is lost. So this functionality simplifies Ubitrack bootstrapping a lot.

## 3.3 Bootstrapping in Ubitrack

For Ubitrack this definition has a special meaning. To integrate new tracking technologies at runtime we need a flexible and adaptable infrastructure. Therefore without loss of generality we assume that these technologies are mobile set-ups which users may bring along. So I redefine the above definition in the more concrete context of Ubitrack:

> *Bootstrapping* is the process of integrating mobile Ubitrack clients into existing systems. The client itself has no a-priori knowledge of its environment, though it brings along its own configuration data. This data can then be used by the environment to incorporate the mobile tracking equipment on the fly and join the two separated Ubitrack system.

The whole bootstrapping process consists of the following steps:

1. Connect to the hitherto unknown ambient infrastructure.
2. Incorporate new sensors at runtime and create new measurements.
3. Merge the separated Spatial Relationship Graph by adding new spatial relationship edges between the separated graphs.

The above mentioned steps are a concrete version of the generic bootstrapping problem. But the single steps themselves also uses bootstrapping mechanisms. This is typical for such problems.

### 3.3.1 Raising Questions and Focus of my work

These steps raise questions on how this can be realized. The most important questions are:

- How can a mobile client get to know about its ambient network infrastructure?
- How can new sensors be incorporated at runtime?
- How can they find out their initial spatial locations to provide new measurements?
- How does the resulting combined Spatial Relationship Graph look like?

In accordance with my general goals in section 1.7, I want to develop a bootstrapping mechanism for Ubitrack and solve the above questions. A first list of possible solutions is:

**Ad-Hoc Networks with Dwarf** Using DWARF already solves the first question. It enables the creation of ad-hoc networks of interdependent services. The ad-hoc network connections are handled by the Service Manager in a suitable way.

**Configuration Exchange to Incorporate New Sensors** To integrate new sensors at runtime, the hard- and software components have to be reconfigured. Therefore the self-describing units have to transfer these descriptions to configure partner components. My focus is to find a model where components of an infrastructure can describe themselves to others outside of it. The distributed character of DWARF allows the design of a dynamic integration of separated Ubitrack systems.

**Finding an Initial Spatial Location** After these components are configured new spatial relationships can be determined because new measurements become possible. This is because of the sensors may track locatable which are hitherto unknown.

**Merging the Spatial Relationship Graph** These new measurements may be used to merge the Spatial Relationship Graph of both separate systems at exactly these graph edges. With this final step the whole integration process is finished and a new, more complex Ubitrack system is formed.

**Out of Scope** The above mentioned questions do not cover the classical issues from Ubicomp sensor networks like energy consumption or communication optimization on network level. First of all, our sensors are not designed to work on low power cells. Although we use a mobile camera for Ubitrack, it is plugged to a notebook which delivers enough energy for our demonstrations. We do not have mobile sensors which rely on internal power cells. Secondly, the communication is handled by DWARF services and the Service Manager. This platform can be adapted to meet to low-level requirements such optimized communication channels or subnet routing.

### 3.3.2 Example

The following simple example should demonstrate the above mentioned steps for a specific tracker: a mobile ARToolKit tracking system.

- A room contains a stationary ARToolKit marker mounted on a wall. Computers in the room run a stationary DWARF based Ubitrack system. The marker delivers its marker pattern for all services which are interested in a specific ability.
- A user wearing a mobile ARToolKit camera connected to a notebook enters a room. The notebook also runs a Ubitrack system.
- The DWARF services responsible for the camera tracking are not started yet because they are not fully configured. The service responsible for the detection of a marker in a video stream needs the pattern description to be able to start.
- The Service Manager s connect to one another. The mobile manager locates the description service with a matching ability for its own detection service. It constructs the service chain and starts all services in that chain: a pose reconstruction and a camera control service for video grabbing.
- The detection service connects itself to the pattern description service and loads the pattern data.

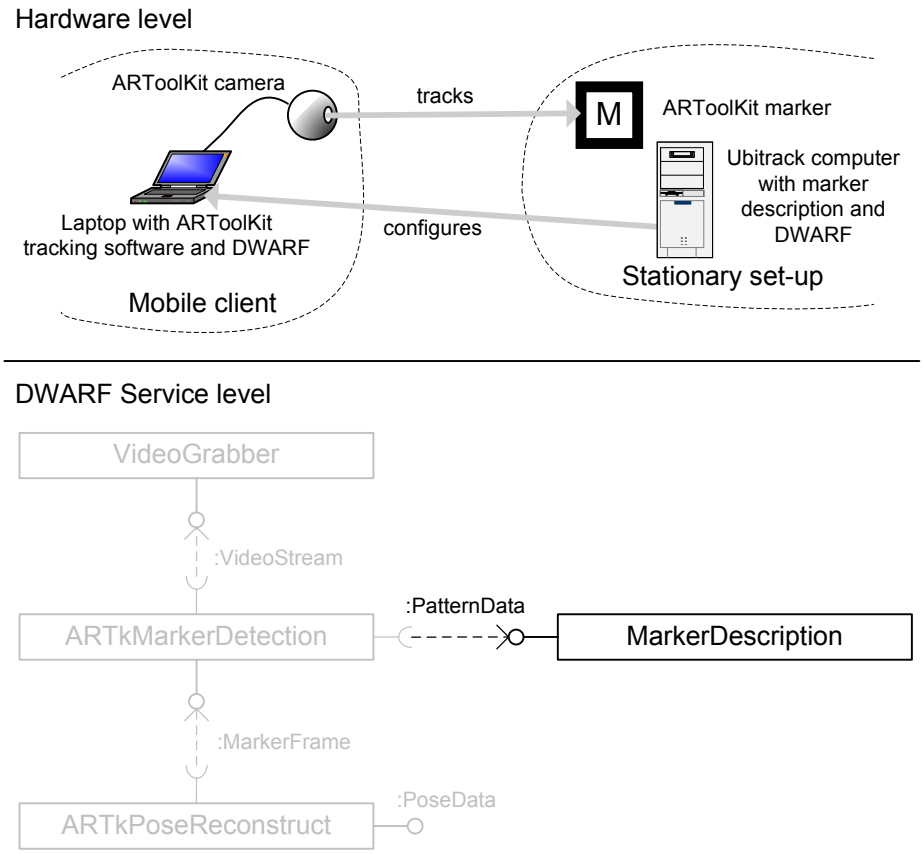



**Figure 3.2:** Simple bootstrapping example for a tracking network based on DWARF. When the `MarkerDescription` is started, the DWARF Service Manager starts the `ARTkMarkerDetection` including all depending services. Therefore if the marker description is available, a new `PoseData` ability is generated.

- The ARToolKit system can now track the marker and gets new measurements. The mobile tracking system is now integrated in the environment.
- Both Spatial Relationship Graphs are merged and updated with the new measurement edge.

This is of course a simplified example, but it demonstrates the basic idea. The example is illustrated in the figure 3.2. It contains only a simplified view. Actually the DWARF Service Manager starts only services when which are needed, i.e. their abilities match to existing needs. The figure shows the opposite: a service with an ability should start one with a need. This can be realized by a second need-ability-relationship: the `ARTkMarkerDetection` service offers a new ability `Startable`. The `MarkerDescription` has the corresponding need. The detection service is started by the Service Manager because of this relationship which can be fulfilled.

The next sections will discuss the three bootstrapping steps in more detail.

## 3.4 Building Ad-Hoc Networks

The first step of the bootstrapping of an Ubitrack system is to build a flexible ad-hoc network. As mentioned this is realized by DWARF and its Service Manager. Services are connected and disconnected spontaneously on infrastructure changes as the manager is aware of the current topology. Finding better ways for creating flexible networks would be out of scope of this diploma thesis.

## 3.5 Exchange of Configuration Data

After the network connections are enabled the sensors have to be reconfigured for the new infrastructure. Ubitrack hardware descriptions deliver the configuration information for the sensors.

### 3.5.1 Ubitrack Hardware Descriptions

Configuration data in Ubitrack is stored in *hardware descriptions*. One description may exist for every node in the Spatial Relationship Graph at the most. When exchanging these descriptions, the sensor hardware gets configured and tracking is enabled. In general, the configuration may include any kind of information which is needed by the software. In Ubitrack following fields may be reasonable:

1. The only necessary field is the object identifier which is used in the SR graph. This identifier must be unique because a graph search relies on unique specifiers. Sensors and locatables both have object IDs.
2. Optional information about hardware characteristics may be included. A sensible distinction must be made between sensors and locatables.
   - The sensors' descriptions may include update rate, camera parameters or what kind of information they may deliver.
   - If the locatable can be used in marker-based tracking, the description may contain pre-created information about the optical features of the locatable.

In DWARF these description are realized by *hardware services* which provide the configuration on demand (see chapter 5).

### 3.5.2 Problems when Exchanging in Large-Scale Networks

Current DWARF application already use this method. Since every measurement and every node is modeled by a separate service, the overall system includes a large amount of services. There are several problems when many services have to be managed.

**Service Manager Scalability** The DWARF Service Manager currently does not scale well with a large number of services. This is due to certain internal facts. First of all the location phase of the manager is not optimized for a large network and as a result not optimized for dealing with a lot of services at the same time. By using the SLP protocol [24] without central

SLP Directory Agents (DA) the detection of changes in the service structures is propagated very slow.

**Restrict the Exchange of Configuration Information** In theory, marker descriptions in an Ubitrack system have to be transfered in advance to the mobile tracking system because the system does not know a-priori what marker are tracked. If there is only a flat infrastructure topology all marker descriptions have to be exchanged. To minimize the needed configuration data, e.g. for sensor configuration, Ubitrack should only transfer relevant data.

**Hierarchical Object Identifiers** On the technical side, object identifiers for the object nodes in the Spatial Relationship Graph must be unique because they must be distinguishable. If you divide the graph into different parts, you can reuse identifiers and construct global IDs by using hierarchical namespaces. A possible approach is the usage of Uniform Resource Identifiers (URI) [8] for Spatial Relationship Graph nodes.

### 3.5.3 Solution: Dividing by Location Context

A possible approach is to divide the Spatial Relationship Graph into smaller pieces and map these pieces to the DWARF service structure. For Ubitrack, a possible way to divide is by spatial neighborhood. The formal Ubiquitous Tracking model introduced supernodes for that purpose.

**Supernodes** The formal Ubiquitous Tracking model groups objects which are "near" each other into *supernodes*. They can be seen as a coarsening of the Spatial Relationship Graph. If a rough estimation is sufficient, Ubitrack searches for spatial relations between supernodes. For example, you can group all objects of a room into a room supernode. This enables high-level queries and answers, e.g. "Where is Bob? – Bob is in the AR lab."

**Location Context** This high-level information can be modeled with *location contexts*. Every supernode represents one specific context in which the object currently is. Nodes can be part of more than one supernode and overlapping of contexts is allowed. You can visualize these relations by a 3-D graph (figure 3.3). The supernodes of the Spatial Relationship Graph are not drawn in the same plane. They form a hierarchical context structure where all nodes in the SR graph are at the bottom level. Overlapping supernodes have common children, so object nodes may be part of more location contexts at the same time.

As you can see in the 3-D graph because objects which are in the same location context and therefore near to each other may not be "near" in the Spatial Relationship Graph. The example shows that the nodes $B$ and $E$ are in the same context: *AR lab*. They are not directly connected by a measurement edge. So both objects are not in the direct neighborhood in terms of graph neighborhood. The top most level is called "*nowhere*". It indicates that no location context is currently set. All nodes in the Spatial Relationship Graph belong to this node. This is the default at start-up when no initial context is estimated.

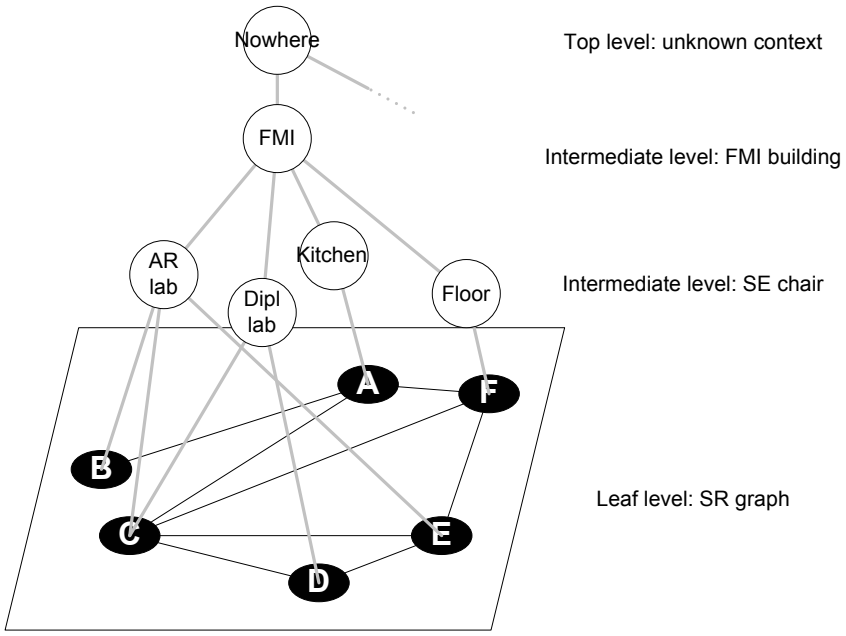



**Figure 3.3:** Visualization of the location context in a Spatial Relationship Graph: The base plane at the bottom level contains the Spatial Relationship Graph; on top of the plane the hierarchical context graph. Overlapping of supernodes is allowed because objects can be part of more supernodes: Object node *C* is part of *AR lab* and *Dipl lab*.

### 3.5.4 Estimation of Initial Context

The problem is that mobile Ubitrack clients have to determine its initial location context by external resources because they cannot manage it alone. If a mobile Ubitrack client is turned on the initial location context is unknown. In this state the system has to exchange all configuration information, too. So the client has to determine the initial context for a first level of restriction. Since the client is totally unaware and therefore not configured for the environment, it is necessary that certain configuration information are exchanged.

Managing a location context for mobile clients is very difficult. Only stationary set-ups have a well defined location context which does not change over time. So they are responsible for setting the correct initial context for mobile clients. The problem is that the mobile context has to be generated by using the infrastructure itself. There are several approaches to generate location contexts for mobile Ubitrack systems. A good overview of the problems can be found in [12].

#### First Approach: Location Context Database

The Ubitrack client must find such a rough initial context because this reduces the amount of information which has to be transfered in order to set up the tracking systems. The easiest way to find such an initial context is to use information which is already available, such as the network configuration.

Because I assumed that the Ubitrack client has already a network connection to its environment it can query a *network location database* where possible mappings of networks to locations are performed. The Ubitrack client gets the network hardware address (MAC ad-

dress) of the network node with which it is connected and sends it to the database. If the MAC address of the node is found the finest location context is returned to the client (see for example table 3.1). After this context is returned, only the configuration which is relevant is exchanged.

The simplest mapping is wireless LAN. Since access points may be mounted statically, the exact location is known. When an access is registered at the access point, the location of the client can be estimated. If more access points are involved, then the exact position can be determined.

**Advantages** The advantages of this approach are that the hierarchical structure can be implemented by directed trees which are efficient in traversing and searching for a object node at the bottom level.

Another advantage is that the structure in independent from the implementation of the SR graph since it is separated.

**Disadvantages** This is also the greatest disadvantage: the management of a data structure is separated from the SR graph and therefore contains parallel information.

Another problem is the definition of a suitable context hierarchy which cannot always be performed at runtime. It has to be maintained manually when the network infrastructure changes.

#### Alternative Approach: Neighborhood Queries

To avoid a second data structure you can return context information as a very imprecise measurement from the Ubitrack system. After the client has access to a network node, a new SR graph edge between the Ubitrack client (regarded as one supernode) and a *context node* on the stationary side is created. The measurement between them is rather imprecise, however neighborhood queries are possible. These queries return all objects which are in the proximity of the Ubitrack client. In figure 3.3 these measurements are between the objects nodes in the plane of the Spatial Relationship Graph and the context nodes in the context graph.

**Advantages** This approach avoids the need for a second hierarchical context structure since no explicit high-level context is used; only Ubitrack measurements are taken into consideration. So we have one framework which does the work.

This approach also does another favor.The location context may be updated by the system at runtime because there is no static database. If the mobile client exits a certain location

| Network MAC address | Type | Location |
|---|---|---|
| `00:02:37:0A:BC:85` | Fixed router | FMI, Lehrstuhl Brügge |
| `00:02:37:0A:BC:86` | Fixed router | FMI, Lehrstuhl Brügge, AR lab |
| `00:12:34:56:78:90` | WLAN access point | FMI, main hall, northeast |
| `00:0C:A7:14:85:3F` | WLAN access point | FMI, library, 3rd floor |

**Table 3.1:** A network location database which maps MAC addresses to location contexts

context to a higher, more coarse level, Ubitrack notices the changes because neighborhood queries deliver no suitable results anymore.

**Disadvantages** However there are trade-offs one has to consider. First of all it is not clear how to model context nodes in the SR graph. Are they supernodes or real pseudo-objects? Further investigations are necessary to find suitable enhancements for the SR graph.

Another problem is that the current Ubitrack implementation cannot deal with this kind of information because the current design inhibits the correct modeling of high-level context information at the communication level (see chapter 5 for details).

## 3.6 Graph Merging

After the initial context is estimated and the trackers are configured, measurements become possible between the mobile Ubitrack client and its environment by finding edges which connect both distinct SR graphs and merge them to one. This is the last step of the Ubitrack bootstrapping problem: merging of two separated Spatial Relationship Graph. How the merging of the graphs is realized depends on how and where the graph is stored. Updating the graph in our distributed approach may become difficult.

## 3.7 Summary

This chapter introduced the high-level concepts of bootstrapping a scalable and distributed Ubitrack system. The next chapter shows the application of these concepts in scenarios where two Ubitracksystems get in contact.

# 4 Mobile Scenarios

**The bootstrapping concepts are applied to mobile scenarios with two Ubitrack systems.**

---

This chapter gives an overview of the smallest scenarios where two Ubitrack systems get in contact with each other. By combining the systems together, the Spatial Relationship Graphs merges and both systems are integrated. This list of cases is complete in this granularity: every more complex situations can be built by combining cases together. For every case I present a description of the interaction of the components in the Ubitrack system.

The purpose of the scenarios is to generate simple use cases for the Ubitrack developer. Boundary conditions are defined by assumptions which are listed in the first section of this chapter. The detailed cases discuss problems and possible solutions for the implementation of such a system. Parts of this chapter are implemented and can be found in chapter 5.

## 4.1 Assumptions for the Scenarios

I will focus on the integration of mobile sensors for Ubitrack. Therefore I will look for examples where a mobile and another mobile or stationary tracking system are merging together. To simplify, I will assume certain facts on which my scenarios are based.

- I distinguish between the user's *mobile client* which consists of mobile tracking equipment and the *stationary set-up* which provides the ambient infrastructure. The mobile client runs a local Ubitrack system with its local Spatial Relationship Graph. All objects which should be tracked by the environment are modeled in the graph.
- The current location context should be determined already. So every object identifier in both Spatial Relationship Graphs is unique and only relevant configuration data is exchanged. Therefore the configuration data can be assigned clearly to only one sensor or locatable.
- The stationary objects in the environment provide configuration information. So every locatable in the environment can describe itself to sensors and every sensor in the environment can be configured by locatable descriptions.
- The client may provide configuration data for its locatable, but this is not mandatory.

## 4.2 Mobile Sensor and Stationary Locatable

If the environment is deployed with locatables, they can publish their descriptions for all sensors which are interested in them. When a mobile client connects with its environment,

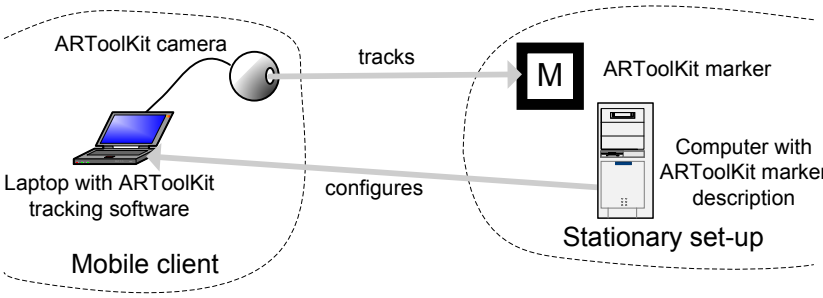



**Figure 4.1:** Mobile ARToolKit camera gets configured by stationary ARToolKit marker descriptions

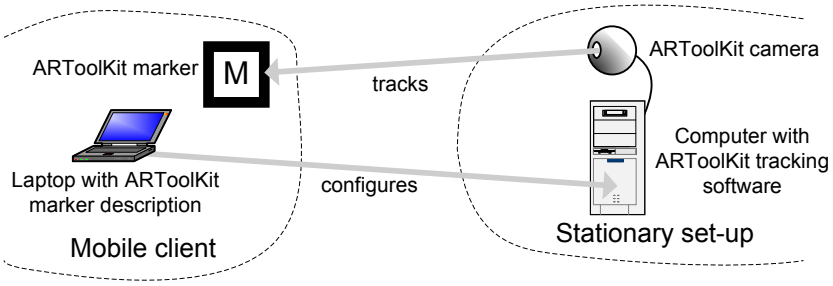



**Figure 4.2:** A mobile ARToolKit marker description configures the stationary ARToolKit camera

these descriptions are transfered to the client which reconfigures its tracking technologies. Therefore, the system recognizes spontaneous changes in the infrastructure, because the new available descriptions may overwrite obsolete configuration data.

Figure 4.1 shows the simple scenario where one ARToolKit tracker connects to a Ubitrack system which consists only of one ARToolKit marker. The configuration contains the marker's object identification (it is marker $M$) and its pattern information. Only when the camera needs the pattern data, it is transfered.

This mechanism is based on the distributed configuration of wide-range trackers as shown in [58]. The mechanism described there is a simplified version without Ubitrack. It was implemented and successfully tested in the ARCHIE project [2] without the use of the formal Ubitrack model.

## 4.3 Mobile Locatable and Stationary Sensor

Now a mobile locatable is to be integrated into the stationary tracking system. As mentioned in the assumptions we can now not longer relay on the availability of a locatable description. So I present two cases: one with and one without description.

### 4.3.1 Describable Locatable

This case is trivial. It is exactly the same case as described above except that mobile client and stationary set-up are reversed (figure 4.2).

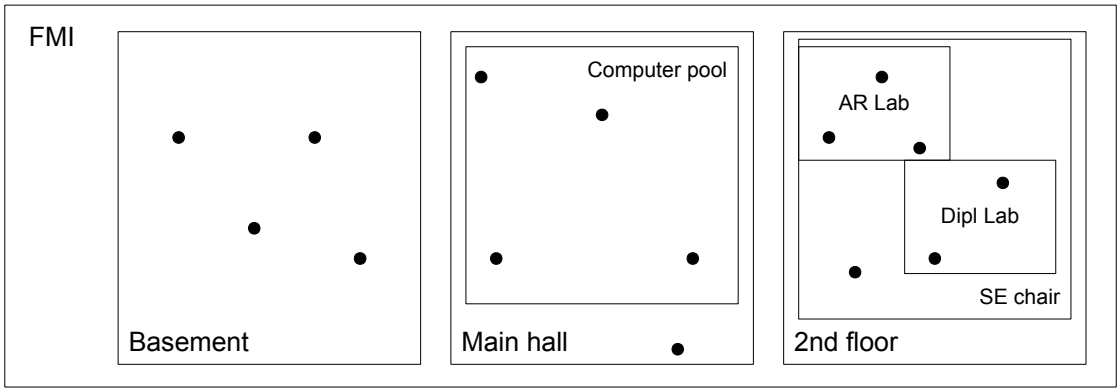



**Figure 4.3:** Example of a spatial index of the FMI building: The rectangles show the partition of the Spatial Relationship Graph nodes (black dots). The classification are physical characteristics of the building itself, i.e. main halls, floors, rooms.

### 4.3.2 Anonymous Locatable

If the locatable does not provide its description, the sensor either cannot track the object at all (in which case Ubitrack is not of great use) or tracks unknown objects. This case occurs whenever a mobile client does not run a local Ubitrack system at all, has no network connection, or simply has no appropriate description of this specific locatable. Furthermore *Marker-less tracking* technologies do not have any descriptions at all.

Thus the stationary sensor creates a so called *anonymous object* in its Spatial Relationship Graph. These nodes have special and unique identifiers but cannot be associated with a real object. The problem is that in reality two nodes with different IDs may be the same object. As long as these anonymous object nodes exist in the graph, Ubitrack cannot merge the two graphs together. So the goal is to find the "real" identifier for anonymous objects. For that purpose, several solutions exists.

#### Neighborhood Queries

In the simplest case, this object is already registered in Ubitrack with another identifier, i.e. there is a second node in the Spatial Relationship Graph which belongs to the same object. This happens when a second path exists between the sensor and the unknown objects over a second sensor. Ubitrack has to query itself for objects which are in the neighborhood of the anonymous object.

Neighborhood queries are best performed on suitable data structures like *spatial indices*. A spatial index is a data structure from database theory. It refers to (or indexes) data by their spatial relationship. You can define these relationships in n-dimensional spaces. Figure 4.3 shows a simple example of a possible allocation of the FMI building. The FMI building is mapped to a rectangle. All sub-rectangles are logical or physical classification of the whole building, such as departments, rooms, and so on.[1] For Ubitrack it is reasonable to group all stationary objects in a spatial index. If a sensor tracks an object, Ubitrack adds the object to its neighborhood by updating the spatial index.

[1] As mentioned in section 3.5.4 this database can also be used to determine the objects which should transfer their configuration data to unconfigured mobile clients.

A suitable data structure for such indices are R-trees [23] or R*-Trees [7] for 3-dimensional partitions. A highly distributed version can be used, e.g. *peer trees* [16]. They are designed to allow neighborhood queries in peer-to-peer networks because they are working on distributed information.

#### Generating Hardware Descriptions

There are tracking technologies which can generate descriptions from hitherto unknown locatables. For example, the ARToolKit contains a configuration program[2] to extract marker patterns from the video stream when the camera sees the marker. Normally, this utility is used for offline generation of the pattern data, however Ubitrack can use this tool to generate the marker patterns online for all unknown ARToolKit markers.

Another application area is marker-less tracking. Since it cannot rely on pre-generated description of the features, it has to generate and save them at runtime. Therefore it can compare the current information with the generated one and if a similarity is found the anonymous object is identified.

#### Motion Pattern Analysis

I concentrated on this method for identifying of anonymous objects. The main principle is to use dynamic quantities to determine new relationships between nodes in the Spatial Relationship Graph. We cannot use absolute values such as position because we do not have a common reference frame. So we use quantities which are independent from both frames, e.g speed or angular velocity. I concentrated on those two quantities because they can be computed easily from the data we receive from Ubitrack, which are position and orientation in arbitrary coordinate systems. By evaluating these values, Ubitrack can determine similar motion behaviors and can infer static relationships between objects from these estimations. Methods for deducing new relationships are presented in chapter 6.

## 4.4 Two Mobile Sensors

The last case is that two mobile tracking systems meet "on the green field" and track each others' locatables. The local relationships between the sensor and the locatable on both systems are unknown, so we cannot assume that the locatable or the tracker are statically attached. Only the tracked relationships are known. A possible set-up is shown in figure 4.4: two sensors track the locatables of each other. The goal of Ubitrack in this case is to find relationships between objects where no actual measurement is available. They amend the merged Spatial Relationship Graph where the methods in section 4.2 and 4.3 can be reapplied. However, this cannot be guaranteed as this section clarifies.

### 4.4.1 Incomplete Spatial Relationship Graphs

The merging of two mobile clients is complicated since they do not need to manage a complete local Spatial Relationship Graph. Ubitrack only knows that there are relationships

[2] The program is called `mk_patt` and is located in the `util/mk_patt` directory of the ARToolKit source code tarball

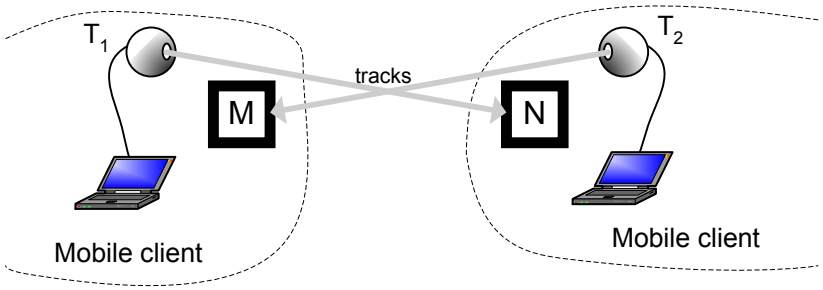



**Figure 4.4:** Two mobile set-ups track each other

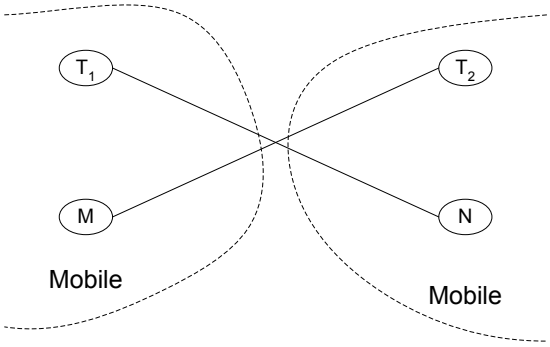



**Figure 4.5:** Starting position: The two sensors $T_1$ and $T_2$ track the locatables $M$ and $N$

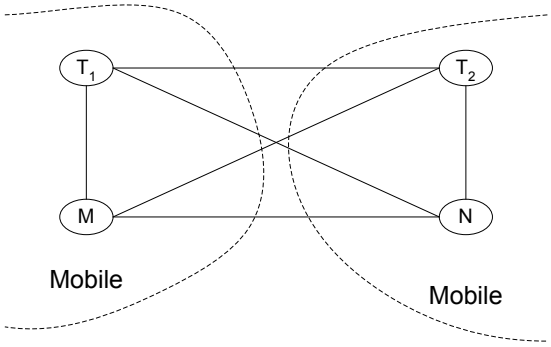



**Figure 4.6:** The desired Spatial Relationship Graph with a complete knowledge between the two systems

between the sensors and the trackers as shown in figure 4.5. Although the graph connects the object on the other node, this is not a whole merging because it contains no relationships between the local sensor and the local locatable. The desired situation for a merged Ubitrack system is shown in figure 4.6. We have to infer new relationships for which no measurements are available.

Whether this goal can be reached cannot be guaranteed here. Even when we find new static relationships, Ubitrack cannot be sure where exactly they are in the Spatial Relationship Graph. Nevertheless this information can be used to increase the accuracy of other measurements even in cases with only marginal knowledge.

### 4.4.2 Describable Locatables

If we assume that both mobile clients deliver locatables descriptions, Ubitrack constructs the graph in figure 4.5. To build a complete Spatial Relationship Graph the current information is not sufficient to infer new relationships. As mentioned before, motion pattern analysis can be used to identify anonymous objects. But furthermore, this method can be used to infer new, hitherto unknown static relationships. For that purpose, Ubitrack investigates the two measurements and tries to find similarities in the motion patterns. "Similar" means that locatable $M$ in respect to sensor $T_2$ moves in the same way as $N$ moves in respect to $T_1$. These is an indication of static relationships somewhere in the complete graph. However, if there are no similarities, Ubitrack cannot infer any new relationships.

If there are similarities, Ubitrack can assume that there are static spatial relationships be-

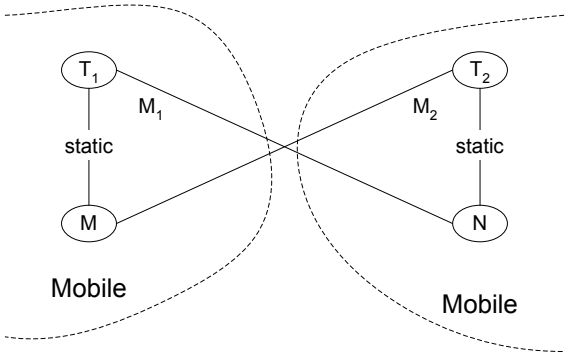



**Figure 4.7:** The mobile sensor is statically attached to the mobile locatable

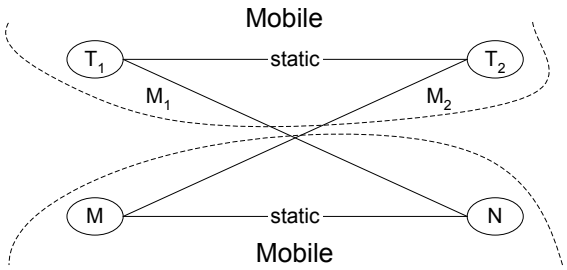



**Figure 4.8:** The two mobile sensors are statically attached, as well as the two mobile locatables

tween objects in the complete graph in figure 4.6. It can therefore hypothesize which relationships may be static: between the sensor and the locatable of the local client ($T_1 \rightarrow M$ and $T_2 \rightarrow N$) or between the sensors and locatables between the local clients ($T_1 \rightarrow T_2$ and $M \rightarrow N$).

**Hypothesis 1:** $T_1 \rightarrow M$ and $T_2 \rightarrow N$ are static
This is the intuitive case: the mobile client consists of a sensor which is statically linked with a locatable (figure 4.7).

**Hypothesis 2:** $T_1 \rightarrow T_2$ and $M \rightarrow N$ are static
This case needs further explanation. As you can see in figure 4.8 the static relationships are between the sensors on the one hand and between the locatables on the other hand. Because the motion pattern analysis only results, that the objects move similarily without knowing the exact static relationships, this case may appear. This is sensible, as the mobile client does not have any knowledge how the locatable is correlated with the sensor. The corresponding set-up is shown in figure 4.9.

The estimation which case is the correct is difficult even with additional information. But in both cases, Ubitrack can use is the fact that static relationships do exist. They can be used to improve the actual measurements. The results of measurement $T_1 \rightarrow N$ can be used to improve the accuracy of $T_2 \rightarrow M$. The Spatial Relationship Graphs in the figures 4.7 and 4.8 show that if locatable $M$ is moving similar to $N$ in their respective sensors' coordinate systems, the graph contains two paths from one sensor to one locatable. The second path may increase the quality of the measurement [10, 48].

### 4.4.3 Anonymous Locatables

Additionally, we cannot rely on hardware descriptions on both mobile clients so the systems initially tracks anonymous locatables. Therefore, the graph in figure 4.5 is not constructed in the shown way by Ubitrack in the first place. The absence of the configuration leads to the creation of anonymous objects. Ubitrack constructs an intermediate graph containing the unidentifiable nodes which contains two new anonymous objects $X$ and $Y$ (figure 4.10). The actual measurement refers to the sensors and $X$ and $Y$ and not the real locatables $M$ and $N$.

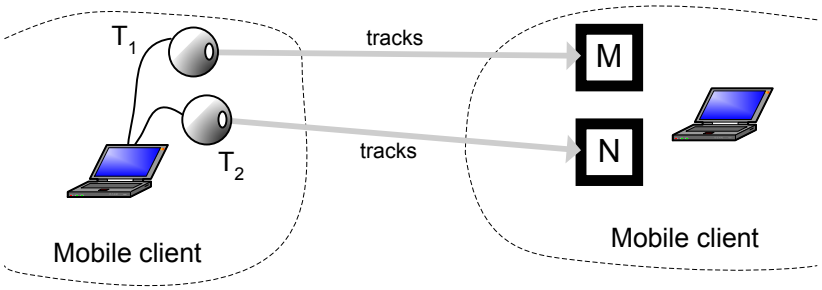



**Figure 4.9:** Second possible set-up with two mobile sensors

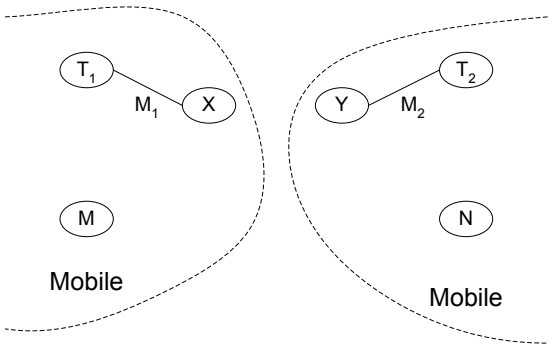



**Figure 4.10:** The real constructed Spatial Relationship Graph containing anonymous objects $X$ and $Y$: The actual measurements $M_i$ are not between the sensors $T_i$ and the real locatables $M$ and $N$, but between the sensors and the anonymous objects $X$ and $Y$.

Since the anonymous objects are placeholders for any real identifier there are four cases which have to be considered. Every anonymous object can be a real identifier given the possibility that the local mobile client can track its own locatable. Therefore the four cases are:

1. $X = M$ and $Y = N$
2. $X = N$ and $Y = M$
3. $X = M$ and $Y = M$
4. $X = N$ and $Y = N$

Cases 1 and 2 are the standard cases: two sensor track two locatables. The anonymous objects must be mapped to all possible combinations of real locatables. Cases 3 and 4 occur when both sensors track the same locatable. The graph will consist of four objects whereas in the real world only three objects exist. For all the four cases the hypotheses 1 and 2 have to be considered, so that there are 8 overall case distinctions. Identifying anonymous objects without real measurements is very difficult. Because this is out of scope of this thesis, this issue is listed for future work (see section 8.2).

## 4.5 Summary

To summarize this chapter: objects can be identified by using their hardware descriptions. These descriptions are transfered from the locatables to the sensors when they are needed. Chapter 5 shows the implementation details for this case in DWARF.

When no identification is provided by the locatables, Ubitrack can estimate this information by investigating the changes in the spatial relationships of objects. These methods are shown in chapter 6.

# 5 Dwarf Implementation of Ubitrack

**Concepts for a scalable implementation of parts of the Ubitrack bootstrapping scenarios by exchanging configuration information.**

---

This chapter overviews the concepts on how Ubitrack bootstrapping can be realized in DWARF using hardware descriptions. It describes methods how this technique may be realized by using descriptions to configure tracking services. The main goal is to stay away from static *PoseData* abilities by exchanging hardware descriptions for a given current location context.

## 5.1 Motivation

The essential way of starting Ubitrack systems is to provide the configuration data for the tracking technologies. As mentioned in chapter 3, three steps have to be conducted for setting up a dynamic sensor network:

1. Connect to a hitherto unknown ambient infrastructure.
2. Incorporate new sensors at runtime and create new measurements.
3. Merge the separated Spatial Relationship Graph by adding new spatial relationship edges between the separated graphs.

This chapter will focus on an DWARF implementation of Ubitrack where the focus of the work is on steps 2 and 3 as step 1 is already realized with the DWARF Service Manager in a suitable way.

I introduce a concept of using *hardware description services* to configure *tracking services*, which are delivering the actual measurements. In terms of the Spatial Relationship Graph, hardware descriptions are the representation of object nodes while tracking services are edges. This concept is qualified to allow new hardware to be added or exchanged at runtime. Step 3 is realized by using Ubitrack Middleware Agents because they are already managing local versions of the Spatial Relationship Graph.

This chapter will also deal with the issues of deploying hardware descriptions in large-scale sensor networks. By using the hierarchical location context concepts in the UMAs, hitherto unconfigured mobile Ubitrack clients are integrated more efficiently.

## 5.2 Definitions

Before the implementation details are presented, this section will give a short list of definitions which are essential for understanding this chapter. These definitions will refine the

rough definitions in chapter 4.

**Mobile Client** As already mentioned the user brings along its own local version of a mobile Ubitrack system. It may consist of one or more mobile sensors which are managed by corresponding UMAs. They are responsible for the distributed graph search and for the estimation of the location context. The UMAs are running in a *mobile mode* which means that the location context of the mobile client is not fixed and therefore has to be set whenever possible. For simplify this implementation, I assume that the mobile client consists of only one UMA. This restriction is necessary for the first design.

**Stationary set-up** The environment consists of stationary set-ups. It may contain more than one UMA which form a *UMA overlay network*. Because the UMAs are stationary, the location context is fixed and can therefore may used to set the context of mobile clients. Whenever mobile clients connects to the stationary set-up, the location context will overwritten on the client's side.

## 5.3 The Context-Aware Ubitrack Middleware Agent

The basis of the distributed implementation of Ubitrack is the Ubitrack Middleware Agent. The UMAs manage the spatial relationships and implement the distributed graph search. They extend the functionality of the DWARF Service Manager to meet the requirements of Ubitrack. Every UMA is running on a network node which manages sensor hardware or locatables. By parsing the local DWARF XML Service Descriptions, the UMA creates the local Spatial Relationship Graph at start-up:

1. For every hardware description, the UMA creates a new object node in the graph.
2. For every local tracking service, the attributes `UTSource` and `UTTarget` are evaluated. If the objects are already created in step 1, the edge is added between the nodes. Otherwise, these objects are considered remote and managed by another UMA.
3. For every remote object, the UMA contacts all other UMAs in its own location context. It then retrieves the hardware descriptions and adds the new edge.

Every UMA must be aware of its own location context. If the UMA is stationary, the location context will be fixed and may be set when the system is started. If the UMA is mobile, the location context is set by the currently connected stationary set-up if possible. A UMA can only manage sensors or locatables in the same context.

How the location context is modeled in DWARF how UMAs can find others in the same location context and how the mobile context is overwritten is explained next.

### 5.3.1 Modeling Location Context

The hierarchical context information has to be modeled for every object in the Spatial Relationship Graph. For determining the correct location context, the UMA must traverse the context graph (figure 3.3 on page 45) on one possible path from the *nowhere*-root to the desired object node. The context graph can be stored in a spatial index structure as introduced

in figure 4.3 on page 50. By performing a query on the spatial index, all context nodes are returned in the specified 3-D area. The hierarchical location context may be implemented by the UMAs using a distributed spatial index structure [16]. That way, hardware descriptions are mapped to location context, because the UMAs determine the context and set it at their managed description services by a special CORBA interface. Therefore, hardware descriptions become location-aware dynamically which has the advantage that the context information has to be hold in the UMA only.

Every context information may be modeled in current DWARF applications by *context attributes* [37, 53]. They are special service attributes with well-defined names. Important for Ubitrack is the context attribute `location`. This attribute contains the current hierarchical location context of its service.

Attributes in general contain only one value because they are name-value pairs. Storing hierarchical information is difficult but possible. They can be encoded e.g. in URI syntax. For example the following predicate:

```
location=loc:/nowhere/FMI/mainhall/*
```

Has to match the attribute:

```
location=loc:/nowhere/FMI/mainhall/computerpool
```

Since all objects in `computerpool` are also in `mainhall`. Clients with no location context can specify the predicate `location=loc:/nowhere/*` which matches all `location` attributes.

### 5.3.2 UMA Overlay Network

Every stationary UMA has a DWARF ability of type `UMA` for other UMAs which can be used to query for the location attribute. It is reasonable to use the already existing inter-UMA ability proposed in [10]. They form a *UMA overlay network* on top of the DWARF Service Manager network. In stationary environments these connections are relatively static, however new connections from mobile clients must be possible. To specify that a UMA is stationary, it adds a certain attribute to its ability: `type=stationary`.

Every UMA has a need for other UMAs of type `UMA` and provides itself an ability with the same type. The UMAs are interconnected at most by two parallel need-ability-relationships. The predicates for these connections contain the current hierarchical location contexts. UMAs connect to all other UMAs which are at the same context level or below, but not at higher levels because the predicates do not match anymore (figure 5.1).

### 5.3.3 Setting the Location Context at Mobile Clients

As mentioned above, only mobile clients are able to change their contexts in accordance to the information provided by stationary set-ups. If an unconnected client, with its local UMA, gets in range to a stationary UMA network, the location context must be transfered. Therefore the first stationary UMA, which connects to the mobile client, can be used as reference. From then on this specific UMA serves as a *ContextUMA* for the mobile UMA which parses the DWARF attributes of the ContextUMA and overwrites its own `location` attribute accordingly.

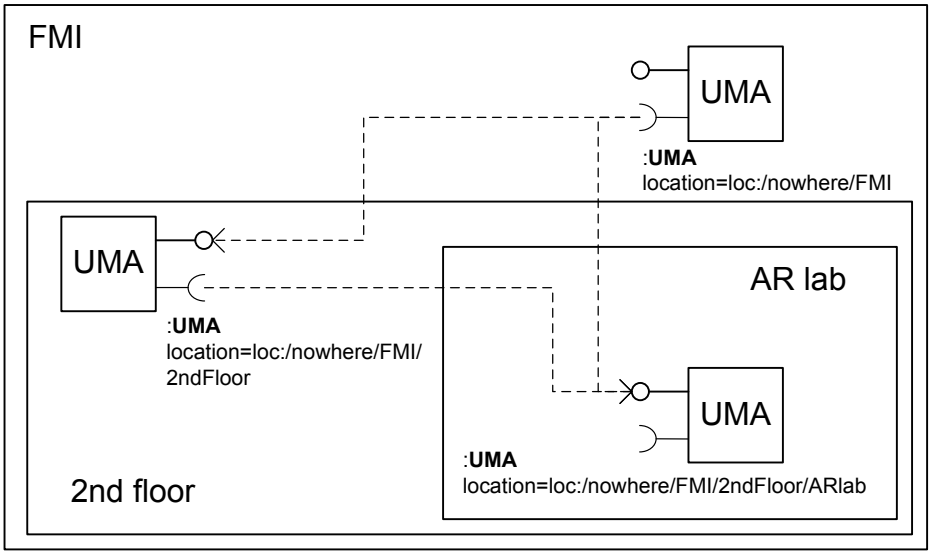



**Figure 5.1:** The UMAs are interconnected by the doubled `UMA` ability-need-relationships. UMAs in higher context levels can connect to those in finer levels, but not vice-versa.

After the context is set, the mobile UMA contacts its local description services and change their `location` attribute. The corresponding object identifiers are updated accordingly. When the connection to the ContextUMA is lost, the context changes back to `loc:/nowhere` which is the default.

## 5.4 Hardware Description Services

Spatial Relationship Graph nodes are the physical objects in the real world; they represent touchable things. These object nodes may be mapped to *hardware description services* which store arbitrary configuration data. In general, there are two kinds of descriptions: *locatable descriptions* and *sensor descriptions*. Both are needed to set up the active tracking services (figure 5.2). The description service offers a `Description` ability for any interested partner. The only mandatory attributes are the type of the description (currently only `locatable` or `sensor`), the class of the hardware (i.e. what tracking hardware is used), and the object identifier deduced from the Spatial Relationship Graph. The identifiers have to be unique in the current location context, as the context is prepended to the ID for the graph search.

### 5.4.1 Implementing the Configuration Information

For implementing the configuration itself, two possible approaches may be used. First, one can model configuration in attributes. The second method is by using a user-defined CORBA interface. Both of them have trade-offs. What implementation is chosen depends on the current requirements:

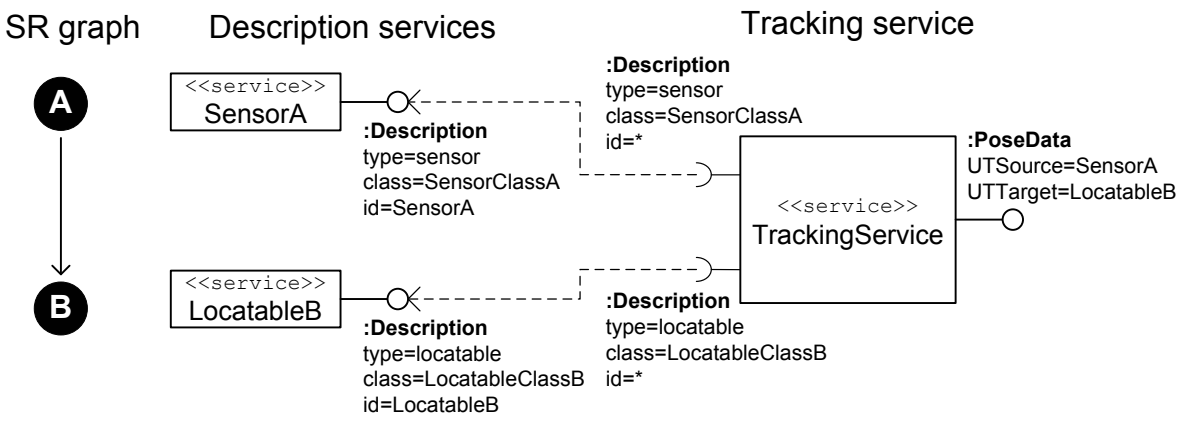



**Figure 5.2:** Sensor and locatable description are merged in the tracking service. The `PoseData` ability corresponds to the given configuration, i.e. `UTSource` and `UTTarget` are set to the values from the description services.

### Attributes and Template Services

The easier method is by specifying configuration information in DWARF attributes. The description service stores all relevant information in arbitrary attributes for the partner service. The partner has to be a template service and therefore is cloned for every distinct description in the system. The clone can configure itself by evaluating its attributes.

**Advantages** The advantage is that no additional programming is needed on the description side. All information is encoded in the XML files. The partner service evaluates the attributes by parsing them e.g. in the session phase.

**Disadvantages** The disadvantage is that template services currently do not scale well with the DWARF Service Manager. For every possible clone a new Service Description is generated within the manager in advance. This results in an explosion of all possible combinations of Service Descriptions. Another problem is that configuration information is, in general, to complex to be modeled in attributes because it may contain binary data. These are difficult to represent as strings.

### Corba Interfaces and Dynamic Abilities

To address the above disadvantages, the description service may implement its own delivery mechanism, e.g. by its own CORBA interface. The ability which offers the interface is generic. No specific information about the configuration is encoded in attributes. The information is encoded in the transferred data of the interface.

Technically, these description services are nothing more than derived versions of the generic `Configuration` service [56]. Currently it may be used as a distributed database for hardware descriptions. The service provides simple CORBA interface for arbitrary back-end databases. The used `ObjrefExporter` connector protocol allows that any service with the appropriate need can import configuration data.

To generate the correct `PoseData` ability at the tracking service, it must evaluate the object identifiers of both needs, grab the configuration by a remote method call, reconfigure

itself, and create the new ability. Therefore the service need not be a template because all adaptations are handled internally.

**Advantages** One advantage is that the configuration data may be specified in a CORBA IDL file and is therefore arbitrary. It is not bound to strings as it is when using attributes. Secondly, the DWARF Service Manager does not have to create additional Service Descriptions since the service need not be a template which reduces the amount of pre-generated Service Descriptions. Therefore it is the preferred choice in large-scale sensor networks.

**Disadvantages** The main disadvantage is that the exchange procedures have to be implemented in the service. If using the `ObjrefExporter` connector, the tracking service has to implement the import of the remote object and must call the method. But it is also responsible for the creation and/or change of the abilities. The description service on the other side must implement the data storage, probably the marshaling of the data, and the delivery.

### 5.4.2 Deployment of Hardware Descriptions

The next step after implementing the description services is their deployment. This problem is a classical field where DWARF meets concepts from traditional Ubiquitous Computing.

**Embedded Descriptions** Under perfect conditions, hardware descriptions are embedded in the devices themselves by wireless connected processing units. If the hardware is replaced or removed, the description is replaced or removed as well. This allows hardware descriptions to be as mobile as their devices. As shown in the SHEEP and ARCHIE demo [52, 2], DWARF is able to run on small wearable systems, currently on HP iPAQs. If DWARF runs on embedded hardware, it will be no problem to integrate services in mobile sensors as well as in locatables.

**Mobile Descriptions in Dwarf** Mobile description services may be realized in DWARF since it provides the suitable mechanism to migrate services to other nodes. The Service Managers update their service lists internally on changes of the locally available service descriptions because they propagate changes via a network broadcast using SLP [24]. Currently this method works for small networks but it is untested for communication across subnets.

When hardware descriptions move to other set-ups, their location context may change. They have to be informed by the UMA if their location context is wrong, so the problem is delegated to the UMA.

### 5.4.3 Examples

The above mentioned concepts are now applied to the commonly used tracking technologies of current DWARF applications. It will map the concepts to the ART system, to ARToolKit, and to Intersense Intertrax.

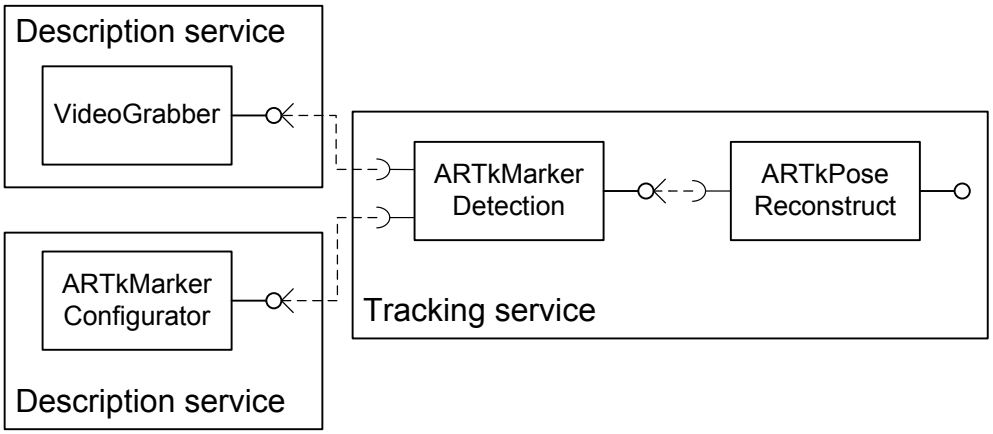



**Figure 5.3:** The ARToolKit subsystem classified in hardware description and tracking services

**ARToolKit** The existing ARToolKit tracking subsystem is the one which is already mostly adapted to Ubitrack and the concepts of dynamic descriptions. It consists of four services: three responsible for the tracking and one for the configuration. The design is based on Martin Wagner's approach in [58].

- `VideoGrabber`
  This service is responsible for the communication with the camera. It maps the video stream into a shared memory block which is made accessible by an ability with the `Shmem` connector.

  This service also provides the camera parameters as DWARF attributes because this service is bound to the actual camera. Therefore this service can be seen as a hardware description service. If the camera changes, the `VideoGrabber` has to be adapted, but the changes are then propagated to the dependent services. It uses the concept of attributes to propagate the configuration.

- `ARTkMarkerDetection`
  This part enables the detection of the marker pattern in the video stream from the `VideoGrabber`. Therefore it evaluates the camera parameters and sends the information about the location of the pattern within the image to a 3-D reconstruction service.

  This service has actually two needs: one for the description of the locatable and one for the video stream also containing the hardware description for the camera. Since these parameter attributes are propagated from the ability of the `VideoGrabber` to its need, the `ARTkMarkerDetection` must be a template service: for every camera exits one Service Description.

- `ARTkPoseReconstruct`
  This service is responsible for the creation of 3-D spatial information from the 2-D information given by `ARTkMarkerDetection`. This service provides the actual `PoseData` abilities. In respect of the terminology of Ubitrack the connected chain consisting of the detection and this reconstruction service is the tracking service (figure 5.3).

- `ARTkMarkerConfigurator`
  There is a first implementation of a description service for ARToolKit markers: the

`ARTkMarkerConfigurator` service. It is a testing service to dynamically load marker patterns into the `ARTkMarkerDetection` service. The configurator is a Qt-based GUI tool for preloading selected patterns into the detection component. Because it relies on interactions of the user it is only usable to a certain extent for dynamic Ubitrack systems. However the design concept is the same.

So the only enhancement is to implement a dynamic `ARTkConfigurator` service which is based on Ubitrack. Every pattern file and therefore every known locatable owns its own description service which provides an interface similar to the `ARTkConfigurator` service. This service is located on the network node whose UMA is supposed to be responsible for the locatable.

**ART Tracking System** In terms of the ART tracking system, Ubitrack cannot introduce description services because the functionality is managed by ART's commercial software D-Track. Every locatable, called ART target, has to be pre-calibrated for D-Track. As D-Track does not provide an API for accessing its internal database of target descriptions, the usage of DWARF description services is limited. Mobile clients cannot bring along new target descriptions and load them into the ART system.

ART's cameras are also pre-configured by the software and so there are no sensor descriptions either.

However, the DWARF `ARTTracker` service can already be adapted to the dynamic marker exchange. If the target is calibrated by D-Track it is mapped to an internal ID number. In addition to the object identifier, the locatable description may contain. The tracker's `Description` need currently only evaluates the internal ID from the description service. That way, we have a mapping of internal IDs to the real object identifier for the `UTTarget` attribute.

**Intertrax** The Intertrax inertial tracker is special since it does not use locatables in the common sense. The tracker's measurements are relative to their own motion. Because it is uses its own inertia, the locatable must be a statically attached to the ground. Therefore, the locatables of the Intertrax must be walls, doors or other objects which cannot be moved. We may simulate a locatable by using the object identifier of this reference object as `UTTarget`. We do not need any locatable description service here. Therefore the `PoseData` ability can be statically coded in the XML description.

Nevertheless, a sensor description makes sense, e.g. for configuring the `Intersense` service. An possible value for the configuration would be the serial or USB port where the tracker is attached.

## 5.5 Implementing the Mobile Scenarios

Now that the concepts of hardware descriptions are explained, the following concepts integrate them into the distributed UMAs. In correspondence with chapter 4 the different scenarios are modeled with the use of a DWARF based conception here. We restrict ourselves to the case where all hardware descriptions are available and omit all cases where anonymous objects occur.

As mentioned in section 5.1, the overall steps to set up a Ubitrack systems are:

1. Connect a mobile Ubitrack client to the hitherto unknown ambient infrastructure.
2. Incorporate new sensors at runtime and create new measurements.
3. Merge the separated Spatial Relationship Graph by adding new spatial relationship edges between the separated graphs.

The following implementation concepts maps these steps to the mobile scenarios where descriptions are available. So the concrete steps for the implementation are:

1. Connect the mobile client with the stationary set-up by the DWARF network.
2. Create new measurements by:
   a) Deducing the initial location context for the mobile client.
   b) Exchanging the configuration information by the description services and configuring the tracking services.
   c) Creating a new `PoseData` abilities for the new measurement.
3. Connect both UMAs with each other in order to add a new edge for the graph search.

The following sections present the details for steps 1–3 for a DWARF based implementation.

### Step 1: Connect to the Dwarf Network

This step is already realized by the DWARF Service Manager. Since it handles ad-hoc networks dynamic connects are recognized. Whenever a mobile client gets connected to the subnet area of a DWARF net, the Service Manager connects automatically and exchanges information about the service infrastructure. So this is already implemented and therefore not focus of this work.

### Step 2: Creation of New Measurements

For creating new measurements the sensor must be configured with the relevant information. Therefore the location context must be set at the mobile client.

**Deducing the Initial Location Context** As described in section 5.3, a ContextUMA must be found from which the correct value for the `location` attribute can be evaluated. A mobile client's UMA has the need for any UMA in the new infrastructure. Therefore the UMA must implement its own version of the session interface of the DWARF Service Manager. Here the UMA gets the attributes of all possible partner UMAs. It evaluates the `location` attribute for all partners and overwrites its own attribute accordingly to the most inaccurate one. This is due to the fact that the roughest possible location context in reach must be correct.

It is important to note that if the mobile client consists of more than one UMA, the location context must be propagated between them.

**Configuring the Tracking Services** The next step is the configuration of the tracking services. To inhibit the spontaneous connection between tracking and hardware description service the predicate for the connection must contain the `location` attribute set to the current context. Therefore the tracking services must maintain the same attribute as their corresponding UMAs. So the UMA must contact its local description services and change its attributes whenever its own `location` attribute changes.

**Generating the new `PoseData` ability** If the correct `location` attribute is given, the tracking services connects to the descriptions by DWARF means. For every connected description, the services generate a new `PoseData` ability and the measurement can be started.

### Step 3: Create the Measurement Edge

The current implementation parses the XML description once at start-up. The create new measurement edges for the UMA, it must provide interfaces for adding new edges. It is sufficient to trigger a new scan for all local `PoseData` abilities in order to reset the Spatial Relationship Graph.

## 5.6 Current Implementation Status

The first static demo, presented in chapter 2, shows the key mechanism of Ubitrack however is completely static. The described concepts in this chapter are not fully integrated into the current UMA. Therefore the proposed dynamic bootstrapping is only partially possible.

Starting from the current status, the integration is easy. Most of the work was the adaption of the old tracking subsystem to the concepts of Ubitrack. So the current status of the Ubitrack migration in DWARF is:

- The IDL are updated to the new *PoseData* data structure. This implied changes in the APIs of the commonly used C++ DWARF tracking library.
- The additional query types of the Ubitrack Query API are specified as IDL interfaces (see chapter 2): `UTPoseDataAsyncPush`, `UTPoseDataAsyncPull`, and `UTPoseDataSyncPull`. Currently implemented in C++ is the commonly used event-based `UTPoseDataAsyncPush` interface.
- For allowing the further testing and implementation, a development branch of the current sources is started.

Further implementations of all concepts in this chapter can use the current adaptations for Ubitrack.

# 6 Estimating New Spatial Relationships

**Bootstrapping without configuration is possible: by investigating similarities in motion patterns, new static spatial relationships can be detected without actual measurements.**

---

If locatable descriptions cannot be exchanged, Ubitrack is unable to merge the Spatial Relationship Graphs since we do not have a common frame of reference and thus no common coordinate system. Therefore Ubitrack may try to find a common frame of reference, i.e. it may try to calibrate two tracking systems. If no common coordinate system may be computed, the comparison of relative quantities, such as velocity, are suitable estimators for spatial relations.

I searched for efficient methods which find such relations at runtime and allow online evaluation. The more precise the estimator are the more computing expensive they become. Ubitrack has no strict realtime requirements in respect to connecting two mobile clients, but a reaction time of more than a few seconds is inappropriate for changing tracking systems with a running AR application.

## 6.1 Motivation

The assumption for the following methods is that we have an incomplete Spatial Relationship Graph according to figure 4.5 on page 52: two measurements and we want to know if some of the objects are statically attached to each other.

This chapter describes the mathematical theory how to determine similarities in motion patterns. The knowledge if two objects move in the same fashion may further be used to deduce information about possible relationships where no measurements are available. That is my main topic of this work and results are presented in chapter 7.

## 6.2 Challenges when Comparing Different Measurements

Comparing two measurements of totally unrelated tracking information is very difficult. Sensor calibration tries to find the transformation parameters between two or more tracking technologies. However, Ubitrack tries to avoid manual calibration steps and therefore tries to deduce relationships online. Since we have no knowledge of the exact transformation parameters, direct comparison of position and orientation data is not possible. Without a common coordinate system, absolute values are meaningful, however relative and dynamic values like speed, acceleration, and rotation velocity may directly be compared. So information about the motion of objects may indicate new relationships in the Spatial Relationship Graph.

Furthermore, this estimation of spatial relationships is also difficult because the measurements sometimes are far from being perfect at all. Every pose contains the spatial information and the time when this measurement was computed. Depending on the tracking technology the following issues have to be considered:

- **Accuracy**:
  Every sensor delivers values only in its given accuracy. If you want to compare two measurements, the accuracy have to be considered and corresponding correction mechanisms should be applied. For example, ARToolKit produces outliers which have to be removed by considering the mean values and the standard deviation.
- **Timing**:
  Every spatial information is gathered at a specific time. Since we have a distributed system architecture and no global time synchronization the compared measurements may actually be taken at different times even if the timestamps are equal.
- **Availability**:
  Sensors deliver results when they are available not when they are needed. So it may happen that for a given time when Ubitrack wants to compare the measurements no spatial information is available. Therefore, inter- and extrapolation of measurements are essential.
- **Scaling**:
  Different trackers represent their measurements in different formats and units. If sensors use fixed standard units Ubitrack may transform them to a common basis. However, sometimes the scaling of the measurements depends on external influences, e.g. temperature. So the scaling factors may vary and cannot considered by Ubitrack.

All methods which should find relationships in motion patterns must take these points into account.

## 6.3 Iterative Closest Point (ICP) Algorithm

This attempt tries to find correlations between the absolute values of the spatial information. However, as mentioned above, a direct comparison is not possible. The ICP algorithm [9] can be used to find relations between the 3D data sets.

For that purpose, Ubitrack may plot the two different position data sets in one common 3D coordinate system. So, the motion curve is not related to each other. The ICP algorithm takes data points on one curve and tries to find their corresponding points in the other curve. For every point this is repeated iteratively with increasing precision. If the correspondences are calculated (figure 6.1) the curves can be matched along the correspondence lines. If the overlap is greater than a threshold, Ubitrack may assume that the curves are identical.

Because this is untested it is not sure whether this approach would work. However some conclusions may be made because of the nature of this approach.

### 6.3.1 Advantages

The main advantage is that it works on the actual measured data. No conversion or preprocessing is needed except an interpolation component for a smooth curve.

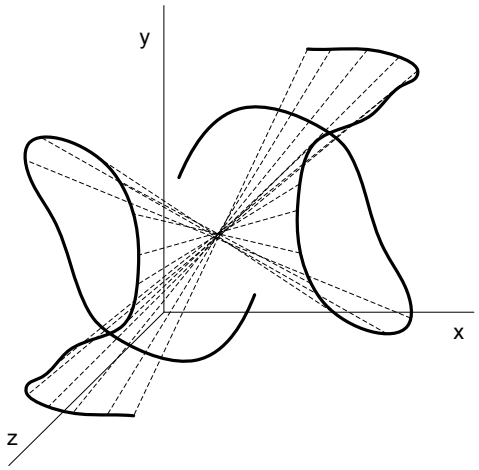



**Figure 6.1:** Example of an ICP based approach for curve fitting

Scaling issues may also be neglected because this is compensated by the ICP itself. The construction of the correspondence lines considers displacements in the data. So if the distance between the points on one curve need not be the same as on the other.

### 6.3.2 Disadvantages

However, the ICP has grave disadvantage: it is a time consuming process to determine every correspondence between the curves. This is actually to slow for the requirements. The estimation need not be computed in realtime, but because of the dynamic nature of AR applications, a reliable estimation have to be fast, not only precise. If the estimation is too slow, the deduced relationship may be not longer accurate. The development of more efficient versions of the algorithm, e.g. [51], may allow faster executions. This has to be investigated in detail in the future.

## 6.4 Computing Speed and Angular Velocity

To be independent from a certain coordinate system the investigation of the objects' velocities may indicate static relationships. To calculate the velocities, Ubitrack must record the current measurements from the tracking services for a specific time period. After that it may calculate the instantaneous velocity for the recorded timestamps. The speed is calculated by using the mean value for all three position components.

### 6.4.1 Speed

When $n$ is the amount of recorded *PoseData* events, Ubitrack can calculate $n-1$ speed values with the following equation:

$$v_i = \sqrt{\frac{\Delta x_i^2 + \Delta y_i^2 + \Delta z_i^2}{\Delta t_i^2}} \quad \text{where} \quad \begin{cases} \Delta x_i = x_{i+1} - x_i \\ \Delta y_i = y_{i+1} - y_i \\ \Delta z_i = z_{i+1} - z_i \\ \Delta t_i = t_{i+1} - t_i \end{cases} \text{and } i = 1, \ldots, n-1 \tag{6.1}$$

### 6.4.2 Angular Velocity

This method may also be used for the angular velocity of the rotation angles. Since the *PoseData* structure stores angles as quaternions, we have to compute the current angle of the rotation. According to [20] every rotation may be expressed by quaternions in the following way:

> Let $q = [\cos(\theta), \sin(\theta)\mathbf{v}]$ and $p = [0, \mathbf{r}]$ be quaternions with $\mathbf{r} = (x, y, z) \in \mathbb{R}^3$. Then $p' = qpq^{-1}$ is $p$ rotated $2\theta$ about the axis $\mathbf{v}$.

From this statement follows directly:

$$q = [w, \sin(\theta)\mathbf{v}] \Rightarrow \theta = 2\arccos(w) \tag{6.2}$$

So to compute the angle we have to consider only the scalar value of a quaternion. However, to get the differential angle between two consecutive values, the difference of the rotation must be computed. Therefore we need the conjugate of a quaternion:

$$p = [w, \mathbf{v}] \Rightarrow p^* = [w, -\mathbf{v}] \tag{6.3}$$

The difference $d$ between a quaternion rotation is computed by the multiplication of the conjugate of the first rotation quaternion $p = [w, \mathbf{v}]$ with the second quaternion $q = [w', \mathbf{v}']$:

$$d = p^*q = [ww' - \mathbf{v} \cdot (-\mathbf{v}'), \mathbf{v} \times (-\mathbf{v}') + w(-\mathbf{v}') + w'\mathbf{v}] \tag{6.4}$$

where $\cdot$ is the scalar product and $\times$ the vector product in $\mathbb{R}^3$.

Since we are only interested in the angle, we only need the scalar value of the product. For the difference of $p = [w, (x, y, z)]$ and $q = [w', (x', y', z')]$ we compute therefore:

$$\Delta w = \left|ww' - (x, y, z) \cdot (-(x', y', z'))\right| = \left|ww' + xx' + yy' + zz'\right| \tag{6.5}$$

The absolute value must be taken since the `arccos` function must calculate the angle in the right quadrant. The values returned are between 0 and $\pi/2$. From equations 6.2 and 6.5 follows:

$$\Delta\theta = 2\arccos\left(\left|ww' + xx' + yy' + zz'\right|\right) \tag{6.6}$$

With these equations we may compute $n - 1$ angular velocities for two *PoseData* streams with the $n$ quaternions $q_i$ similar to equation 6.1:

$$\omega_i = \frac{\Delta\theta_i}{\Delta t_i} = \frac{2\arccos\left(|w_{i+1}w_i + x_{i+1}x_i + y_{i+1}y_i + z_{i+1}z_i|\right)}{t_{i+1} - t_i} \quad \text{with } i = 1, \ldots, n-1 \tag{6.7}$$

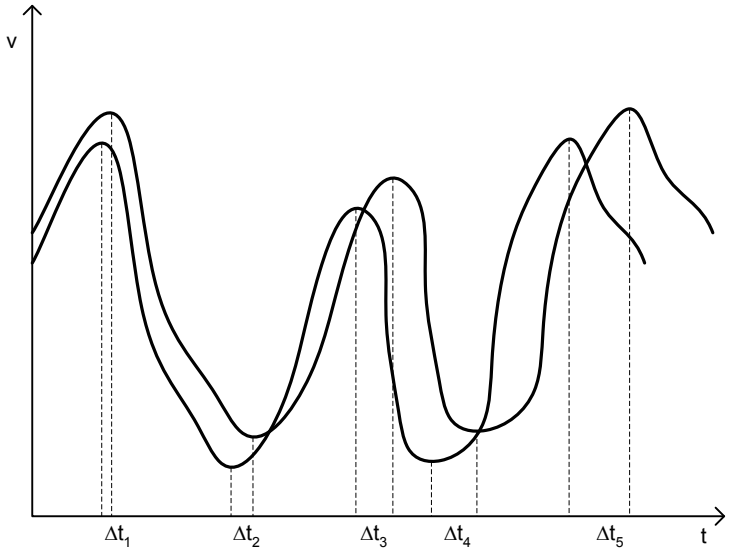



**Figure 6.2:** Comparing the time distance between the extrema of the speed plot

This calculation of the instantaneous velocities does not consider the acceleration between consecutive timestamps: it assumes constant speed. The discrete values $v_i$ and $\omega_i$ can be plotted against the $t_i$ (figure 6.2 shows the curve for the speed).

After calculating the dynamic quantities, Ubitrack may investigate the data sets for similarities. This section presents two different methods for this task: a direct comparison of the data and a frequency-based spectral analysis.

## 6.5 Direct Comparison of Velocity

In order to find similarities you must interpolate the discrete data points since the $v_i$ need not be at the same timestamps $t_i$ in both data sets; this also applies for the $\omega_i$. Now you may compute the extrema of the graph. The solution depends of course on the type of interpolation which is used. With linear interpolation the extrema can be determined by simply looking to the neighbor points. If both are smaller a local maximum will be found, if both are bigger a minimum is found. In higher order interpolations (which are preferred as you will see), the maxima and minima are in the neighborhood of those data points.

If the extrema of both data sets are determined, Ubitrack may compute the time difference of the corresponding time stamps (figure 6.2). In the best case the extrema of both graphs may be found at the same time stamp. But time offsets may occur due to a missing global time synchronization. If the motion is similar, the time differences must be constant over the whole measurement.

As you may see in figure 6.2 these differences are not constant even if the motion is very similar. This can happen when the measurements are drifting during the observation time.

### 6.5.1 Advantages

This method is simple and the speed values may be computed very efficient. If using a linear interpolation for the data sets, the computation of the extrema is a simple comparison step.

### 6.5.2 Disadvantages

However, this is not the preferred choice when analyzing motion patterns. The problem is to find a suitable interpolation for the values. In linear interpolation too many characteristics of the data is not considered and so it may adulterate the results. For example, computing the extrema is very efficient, however it is very inaccurate. The differences between the extrema on the time axis may vary too much.

Another problem is that two exact equal graphs may have computed from complete different source data. This is due to the ambiguity of the absolute velocity: if you investigate the absolute velocity you cannot ascertain which component of the spatial vectors in the *PoseData* has been responsible for the motion. This is a general problem when using quantities which are not absolute in a frame of reference. There are arbitrary motion patterns which are not statically related but produce the same velocity data.

## 6.6 Frequency Analysis of Velocity

To address the disadvantages of a direct comparison, a frequency-based approach can be applied. This method is suitable for estimating static relationships in sensor data which is delivered in a fixed update rate. By investigating the *spectrum* of a motion instead of the original *signal* itself, subtle differences in the movement may result in clear difference in the spectrum. So even small changes may be noticed. My approach is based on the *coherence* function for two spectra similar to the approach in [34]. However in contrast to this approach, we consider velocities rather than acceleration.

### 6.6.1 Introduction in Spectral Analysis

Before I present the actual work in this field, a short introduction will clarify the mathematical concepts of the analysis.

Every time dependent signal may be transformed in the frequency domain by using the *Fourier transform*. It computes the frequency proportions for a given time signal. The result is a function in the frequency domain exactly for the input function. Detailed information can be found, for example, in [11].

#### Power Spectrum

We try to find similarities in the *power spectra* of two signals. A power spectrum visualizes the portion of the signal's power at a certain frequency. It is calculated by the FFT. Let $H(f)$ the transformed function of $h(t)$. Then

$$S(f) = |H(f)|^2 \tag{6.8}$$

is the power spectrum of $h(t)$ [38, 60].

#### Window Functions

As the Fourier transform is working on periodic functions only, the FFT must be optimized for unrepeated velocity signals. Therefore, the input signal has to be divided in separate *windows* which are investigated separately. The mean value of all power spectra is taken as

the result. The more windows are used for an aperiodic function the more precise is the calculation of the Fourier transform. However, computation time is also increasing. Windows may also overlap for more accuracy.

In order to simulate a pseudo-periodic behavior every window is multiplied with a *window function*. This function has the property that it zeros the borders in $h(t)$ but try not to disturb the properties of $H(f)$ at the same time.

For the analysis of a data set with $n$ entries, we used three Hanning windows of length $n/2$ with an overlap of $n/4$ for computing the FFT. The frequency area for the investigation is between 0 – 10Hz according to the nature of human motion [63, 34]. Therefore we need a constant sample rate of at least 20Hz according to the Nyquist-Shannon sampling theorem [44].

#### Coherence and the Similarity Probability

The *coherence* function is the measure for the similarity of two power spectra. It compares two spectra according to their *cross spectrum* and normalizes the result. The cross spectrum is the Fourier transformed function of the *cross correlation* function.

Coherence measures the extent to which two signals are linearly related at each frequency, with 1 indicating that two signals are highly correlated at a given frequency and 0 indicating that two signals are uncorrelated at that frequency [13, 34]. It is computed with:

$$C_{xy}(f) = \frac{|S_{xy}(f)|^2}{S_{xx}(f)S_{yy}(f)} \tag{6.9}$$

where $S_{xy}(f)$ is the cross spectrum and $S_{xx}(f)$ and $S_{yy}(f)$ the power spectra of two signals $x$ and $y$.

From the coherence we may compute a scalar probability value which indicates to similarity with one value. For that purpose, we can multiply a probability density function $d(f)$ to the coherence and integrate it over the whole frequency band:

$$P_{xy} = \int_0^{10} d(f)C_{xy}(f)\,df \qquad \text{with} \int_0^{10} d(f)\,df = 1 \tag{6.10}$$

As proposed in [34] we may assume an equal distribution function for all frequencies and therefore use $d(f) = \frac{1}{10}$ for the density function. Equation 6.10 becomes:

$$P_{xy} = \frac{1}{10}\int_0^{10} C_{xy}(f)\,df \tag{6.11}$$

This formula computes a value between 0 and 1 which stands for the probability that the two measured signals are actually the same.

### 6.6.2 Adopting to Velocity Frequency Analysis

The method in [34] uses accelerometers as sensors. So the investigates coherence function uses the acceleration as time signal. Since Ubitrack's sensors produce spatial data the velocity is the better choice for investigation than acceleration. Fortunately the derivation in the

Fourier transform results only in an additional factor. This factor is canceled down by the fraction of the coherence.

Is $h(t)$ the signal and $H(f)$ the corresponding transformed function then:

$$h(t) \mapsto H(f) \quad \Longrightarrow \quad \frac{d}{dt}h(t) \mapsto ifH(f) \tag{6.12}$$

The corresponding equation for the integration is:

$$\int_{-\infty}^{\infty} h(t)\,dt = 0 \ \wedge \ h(t) \mapsto H(f) \quad \Longrightarrow \quad \int_{-\infty}^{t} h(t)\,dt \mapsto \frac{1}{if}H(f) \tag{6.13}$$

So for the frequency analysis it is irrelevant to measure acceleration or velocity: the coherence does not change its values since all factors are canceled down.

### 6.6.3 Sampling the Measurements

To analyze the discrete measurements, the data has to be sampled at a constant rate. This is due to the nature of the *Discrete Fourier Transform* (DFT). It transforms discrete time signals into the frequency domain. The *Fast Fourier Transform* (FFT) is an efficient implementation of the DFT. However, the DFT relies on constant sample rates which is not always given by our sensors. Therefore, an interpolation step is here also necessary to get measurements in constant time intervals. If a tracker is not able to deliver the results with the required rate of 20Hz then a suitable interpolation component is instantiated.

In order to get the constant sample rate of 20Hz, some measurements have to be interpolated. For example, the `ARTTracker` service can be configured to deliver results for a specific update rate, the ARToolKit system cannot. Here, the interpolation will influence the results of the analysis. For simplicity I used the linear interpolation component of the `Inference` service.

### 6.6.4 Advantages

This approach addresses the disadvantages from the direct comparison above. A frequency analysis reacts on subtle differences in motion pattern better than direct comparison does. As shown in [34] this approach may also be used with low-cost accelerometers to determine static relationships in the Spatial Relationship Graph. They may therefore be used to verify assumptions when future advanced methods try to minimize the amount of anonymous objects.

This method may be used to investigate snapshots of measurements. In general this approach is independent from the length of the data series because the only important factor is the sample rate of 20Hz. However, you may increase the accuracy and therefore the relevance of the probability value by increasing the time interval in which the records are investigated.

### 6.6.5 Disadvantages

The trade-offs of this method is the computation time. It is definitely more computing intensive as the simple velocity comparison. Even when using a FFT based approach, it cannot fulfill “hard” realtime requirements.

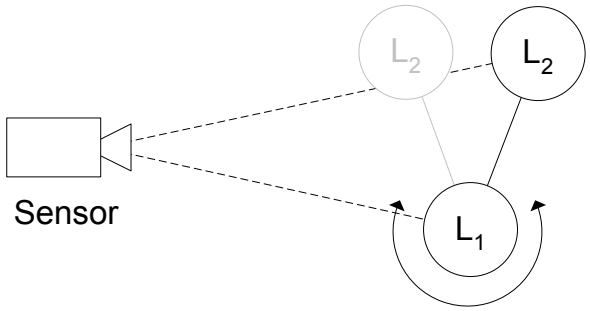



**Figure 6.3:** Hard ascertainable motion pattern

Moreover, this method also depends on the choice of the interpolation function. The suitable interpolation for this method should not disturb the power spectra of the single signals. A frequency based approach for the interpolation should be preferable.

Ambiguities cannot be eliminated by this approach either. A frequency analysis depends from the same velocity data. Objects which are equal in their velocities, but not in their static relationship, produce wrong results. However since this method detects variations in the data better than the presented direct methods, those ambiguities are detected earlier.

## 6.7 Hard Ascertainable Motion Patterns

There are motion patterns which may be hardly detected as similar. Figure 6.3 shows an example for this motion. One locatable stays at the same position when the other is rotated around it. The translation for the first is therefore zero whereas the other shows a periodic movement. However, in this case the angular velocity of both locatables stays similar. So we may expect that the angular velocity is a better quantity to find similarities.

# 7 Results

**Results of an offline frequency-based motion pattern analysis are presented for several test cases. Different tracking technologies are compared to one another.**

---

This chapter presents the results of this thesis. It describes the test set-up and how the test series have been recorded and analyzed. The results are diagrams showing the coherence function and the corresponding probability value.

## 7.1 The Test Cases

For evaluation of the frequency analysis several test cases are conducted using DWARF and the new Ubitrack components. The analysis is performed offline using recorded data from the DWARF set-up. Therefore, MATLAB is used for analyze and visualize the data sets and for computing the coherence.

The conducted test cases are:

1. One ART target and the same data with random noise
2. Two statically attached ART targets
3. Two detached ART targets
4. One ART target and an attached ARToolKit marker
5. One ART target and an attached Intertrax

The ART tracker is fixed mounted in the room. The calibrated ART coordinate system is considered as the global room coordinate system. The tests are conducted in the AR lab. For the test cases 1–3 single ART locatables where used. For the test cases 4 and 5, the "superlocatable" and camera shown in figures 7.1 and 7.2 are used.

For every locatable, a `PoseDataLogger` recorded the measurements in text files. It is a generic DWARF service which receives *PoseData* events by the `UTPoseDataAsyncPush` interface. It was originally written for the ARCHIE demo by Chris Kulas and later adapted for Ubitrack by Daniel Pustka. Every test case consists of two test series for the two locatables. They are recorded in the following MATLAB files:

| | |
|---|---|
| `one_marker.mat` | The recorded data from one ART target is duplicated and random noise is added to the values. |
| `two_static_markers.mat` | Two ART targets are attached statically and the two measurements are imported into this MATLAB file. |

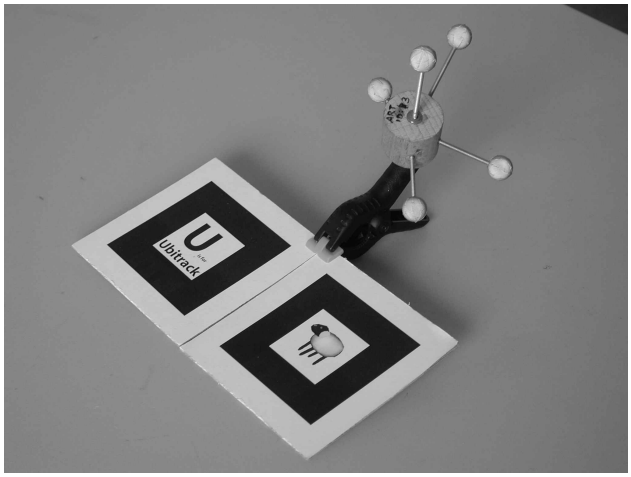



**Figure 7.1:** The designed "superlocatable" for the test set-up


**Figure 7.2:** The used ARToolKit camera used for the test set-up

| | |
|---|---|
| `two_single_markers.mat` | The two markers from test case 2 are now detached and moved separately. |
| `art_artk_markers.mat` | One ART target is attached to one ARToolKit marker. The ARToolKit camera is mounted at a fixed position in the room, so ART and camera are both statically related. This files contains the measurements of the SHEEP marker and the ART target which are shown in figure 7.1. |
| `art_isense.mat` | One ART target is attached to the Intersense Intertrax inertial tracker. Both are statically fixed, so the Intertrax outputs the angular information relative to the ART target. The data is taken from the measurements of the camera's ART target in the Intertrax shown in figure 7.2. |

These files can be found at the following directory in the Linux cluster:

```
/shared/dwarf/maybeusefull/ubitrack/matlab_test_series/
```

The following test cases evaluates the data series. They refer to figures which are listed at the end of this chapter.

## 7.2 Test Case 1: Single ART Target

In the first test case the measurements of a single ART target are recorded as the first series. The second series is computed by adding randomized, normal-distributed noise on the single components. This should result in differences at higher frequencies (above 10Hz) which should not be considered in the frequency analysis. The ART system is configured to send measurements at a rate of 20Hz. This case should test the overall whether the frequency based approach is possible with the data delivered by Ubitrack.

**Analyzing the Speed** Figure 7.4 shows the computed absolute speed of the locatable and its noised version. The noise results in a vibration in a high frequency and therefore deteriorate the data. But the coherence function (figure 7.5) never drops below the value 0.5 which indicates high correspondence in the input signals. Furthermore the probability value is also high with a value of $p = 0.89$. Let us assume that correspondences are recognized at values of about $0.9 \pm 0.1$.

The tolerance value of $10\%$ should be appropriate. Long term evaluations should be conducted to prove or falsify this assumption.

**Analyzing the Angular Velocity** The corresponding analysis of the angular velocity can be found in the figures 7.6 and 7.7. The noise dominates over the original signal. Therefore the corresponding coherence function shows totally uncorrelated signals which is indicated in the heavy jumps of the coherence values at the different frequencies. Therefore the probability values is very low at $p = 0.49$.

## 7.3 Test Case 2: Two Attached ART Targets

Not a combined locatable of two ART targets is moved in the room. Since the ART tracker is one sensor and the target is static, the motion inevitably must be correlated.

**Analyzing the Speed** The speed curve in figure 7.8 shows overall correlation, but there are time intervals where the curve shows differences, for example between 11 and 13 seconds of the measurement. The coherence in figure 7.9 shows tremendous divergence in low frequencies. Low frequencies should be more important than higher ones because constant motion is more likely than rapid changes. The overall probability value is the exact threshold of $p = 0.80$ and therefore in the limits.

The reason for the discrepancy is the motion pattern of figure 6.3 of page 74. The shown movement takes place during the above mentioned phase between 11 and 13 seconds. As you can see this motion pattern disturbs the analysis. The angular velocity should result in a better coherence.

**Analyzing the Angular Velocity** The direct comparison of the velocity curves (figure 7.10) shows great correspondences. As expected the angular velocity shows a high coherence with the probability value $p = 0.95$ (figure 7.11). This is result which completely strengthens the frequency based approach.

## 7.4 Test Case 3: Two Detached ART Targets

Now we detach the two ART targets and move them independently. This should prove that the approach correctly recognizes uncorrelated signals.

**Analyzing the Speed** If one compares the velocities in figure 7.12 then a certain similarity can be found. Since the locatables are detached, it is very unlikely that they were moved similar in the test case. A direct comparison would be difficult in that case. The coherence function (figure 7.13) shows uncorrelated signals in the low frequencies. In high frequencies

the coherence is high. The probability value of $p = 0.72$ is therefore relatively high for an independent signal.

The high frequencies indicate that there is a correspondence. Because it is more likely that motion is in the low frequencies, it can be assumed that no motion took place with high frequencies at all in both series. There is a real correspondence: the missing of motion.

This is result is comparable to test case 2 (figure 7.9). This indicates that either case 2 analyzed an unfavorable motion or the linear speed is not a suitable characteristic for a frequency based analysis.

**Analyzing the Angular Velocity** As the figures 7.14 and 7.15 show there is no similarity in the motion patterns. This result is a clear indication that the movement is independent. It also strengthen the assumption that the angular velocity is more precise quantity for the estimation. The probability value $p = 0.45$ is clearly under the threshold of test case 1.

## 7.5 Test Case 4: ART Target Attached to ARToolKit Marker

This test case uses two different tracking technologies for analyzing the motion pattern. Because the ARToolKit system do not provide the tracking results at a constant update rate, the data has to interpolated by an external component. Instead of using the linear interpolation of the current `Inference` service, I use the `PoseTool` analyzing software and its Kalman Filter based interpolation. The `PoseTool` is a developer tool of Daniel Puskta [48] for applying Kalman Filters to a stream of *PoseData* events. The constant update rate for the filter is according to the analysis 20Hz.

The problems with the data of the ARToolKit is that it contains outliers. This is due to the fact that it may detect markers which are in fact not visible. To remove the outliers, I used a very simple removing algorithm: Let be $m_i$ the measurement of a *PoseData* stream of length $n$. Then

$$\forall m_i \quad \text{if } |m_i - \mu| \geq 3\sigma \text{ then } m_i = m_{i-1} \tag{7.1}$$

where $\mu$ is the mean value and $\sigma$ the standard deviation of the data series. Currently the above equation takes only the position $(x_i, y_i, z_i)$ into account for $m_i$: if one component is outlying then the whole measurement is replaced by the previous one.

**Analyzing the Speed** As you can see in figures, the analysis shows uncorrelated motion patterns. Because the locatables are attached statically this result is definitely wrong. The speed diagram (figure 7.16) shows no correlation at all. For the ARToolKit there are even 3 outlying peaks which have not been filtered by the simple equation 7.1. The coherence (figure 7.17) shows the lowest probability value of all speed test cases: $p = 0.38$.

**Analyzing the Angular Velocity** The angular velocity shows a similar behavior: the motion seems to be not correlated. The direct comparison of the angular velocity diagrams show that the peaks of the ARToolKit are much higher than of the ART target (figure 7.18). Finding outliers in the angular velocity curve is more difficult than in speed cases, because

the rotation angles can flip which results in higher velocity values as normal. For an angular velocity analysis the coherence (figure 7.19) shows totally uncorrelated signals: the portability value $p = 0.35$ is even lower than with synthesized data of the first test case.

## 7.6 Test Case 5: ART Target Attached to Intertrax

Now we compare the test series of the camera's ART target with the attached Intertrax (figure 7.2). Because the Intertrax inertial tracker measures orientation values only. Therefore, this test case only evaluates the angular velocity. As you can see in figure 7.20 the angular velocities show great correlations at all times. The correspondent coherence (figure 7.21) shows a high correlation at low frequencies. For higher frequencies the two signals differ. This is an expected result because signal noise of the sensors can deteriorate correlations in high frequencies. Therefore, the two data sets show a static relationship. The probability value of $p = 0.68$ is below a sensible threshold although the overall coherence curve indicates the contrary.

## 7.7 Summary

The test cases of the frequency based approach shows usable results. It also strengthen the theory that it does not suffice to rely on linear speed only because natural motion of locatables may contain patterns which cannot be detected by simple speed comparison. Therefore the conducted test cases show that the angular velocity is the better choice. Combining both results may be the best choice to ensure that the estimation is correct.

**Interpreting the Results** Test cases 1–3 and 5 showed the overall theory is working for the tracking systems which are available for DWARF. The only exception is the ARToolKit in test case 4. There are influences which might falsify the results:

- Probably the outlier removal used a too simple algorithm for real ARToolKit data. More advanced algorithms should be evaluated. A possible approach would be to filter both, the original spatial data and the calculated velocity data. But computation complexity will increase and online analysis will become more difficult.
- The test set-up was coincidentally prepared that the $y$ component of the position vector pointed to the same direction. Therefore a direct comparison of the components of ARToolKit and ART was possible. As shown in figure 7.3, the ARToolKit shows an offset of several seconds compared with the first peak of the ART. This might be an indication that either the time stamps are old or measurements are inaccurate:
  - Given that the measurements were really taken at the recorded time, then the measurements of ARToolKit are completely inaccurate. There is no correspondence of the peaks in the direct comparison of the velocity. But the ARToolKit is successfully used in many AR applications, so this does not seem to be a plausible answer.
  - If the time stamps of the ARToolKit measurements are older than the actual measurement, the resulting curve is right-shifted compared to correct data. This is the

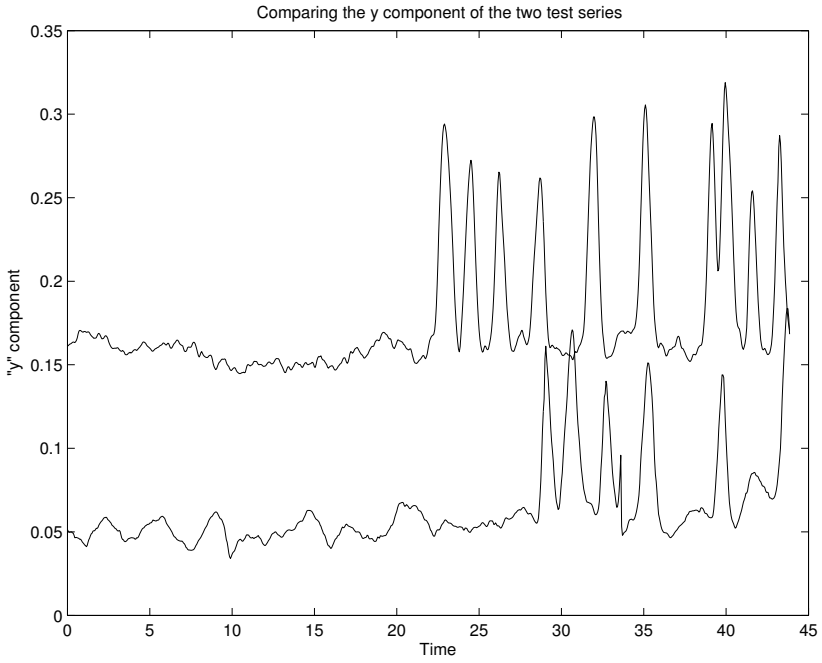



**Figure 7.3:** Comparing the $y$ component of the ART and ARToolKit position vectors: Because the two coordinate systems coincidentally pointed to the same direction, the $y$ component must show related behavior. The ARToolKit values (lower line) shows an offset of several seconds compared with the ART target (top line).

more obvious case. There can be problems within the DWARF implementations of the ARToolKit services and/or timing problems in processing libraries, such as the CORBA Notification Service.

- In the worst case this might be an indication that the ARToolKit might not be useful for this kind of analysis at all. This would be the case that the errors of the measurement influence the frequency analysis in such a way that all frequencies are not related anymore. But this sounds very unlikely.

**Improving the Results** As mentioned, higher frequencies are not as relevant as lower ones because measurement errors are more likely to influence the result in higher frequency bands. Therefore they are not less significant. This observation can be considered when calculating the probability value. Instead of using the direct integral in equation 6.11 on page 72, the coherence function $C_{xy}$ can be multiplied with a density function $d(f)$. With suitable function, high frequencies become less significant than the lower ones. The probability value is then calculated by:

$$p = \int_0^{10} d(f) C_{xy}(f)\, df \tag{7.2}$$

with a suitable probability density function $d(f)$. Although the computation amount is increased, the probability value will become more meaningful.

Results of test case 1: Speed

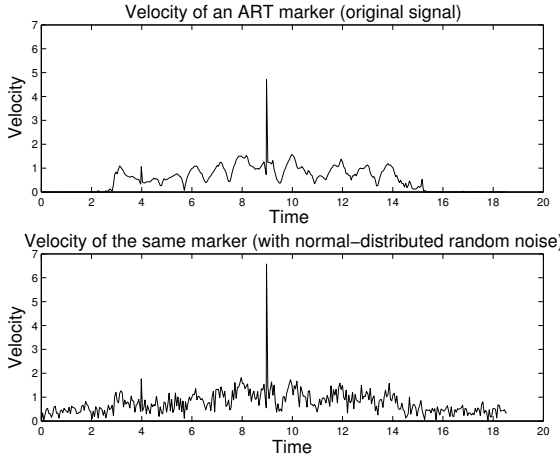



**Figure 7.4:** Speed diagrams

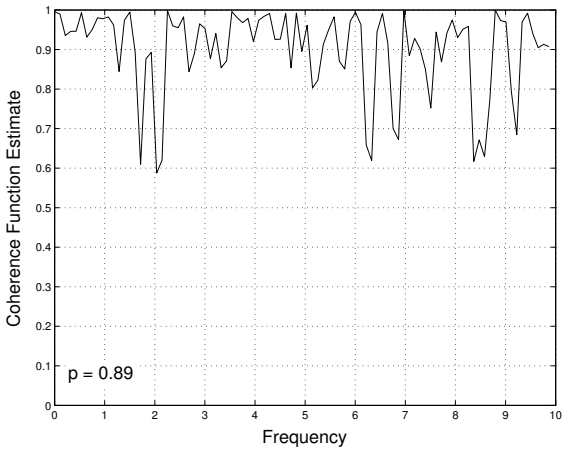



**Figure 7.5:** Coherence function of the speed

Results of test case 1: Angular velocity

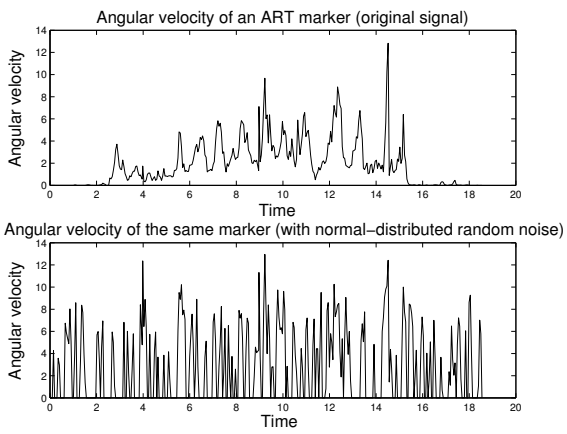



**Figure 7.6:** Angular velocity diagrams

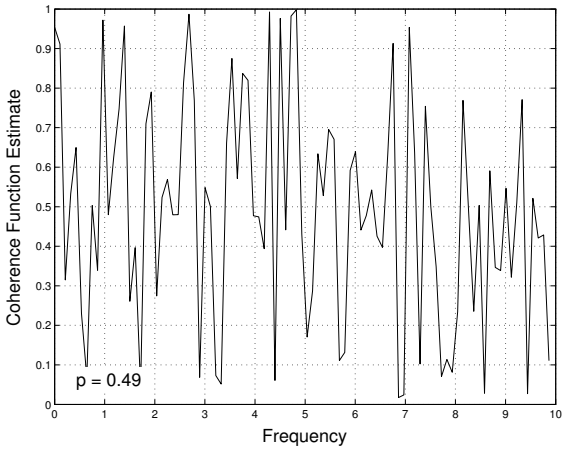



**Figure 7.7:** Coherence function of the angular velocity

Results of test case 2: Speed

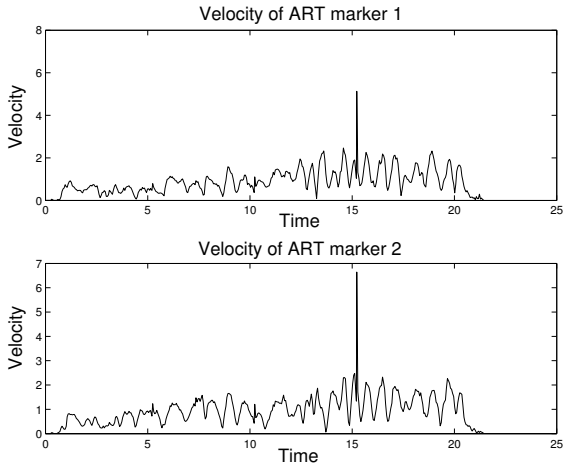



**Figure 7.8:** Speed diagrams

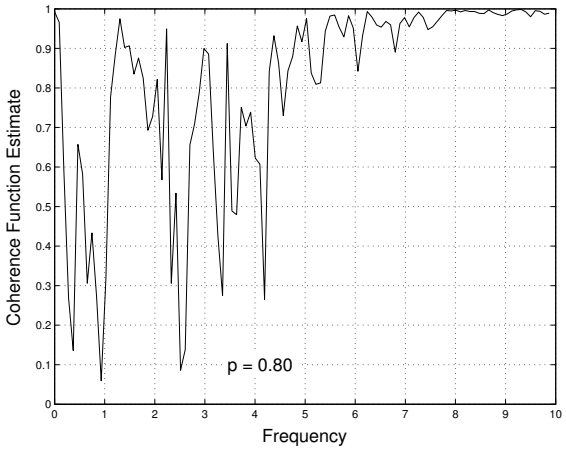



**Figure 7.9:** Coherence function of the speed

Results of test case 2: Angular velocity

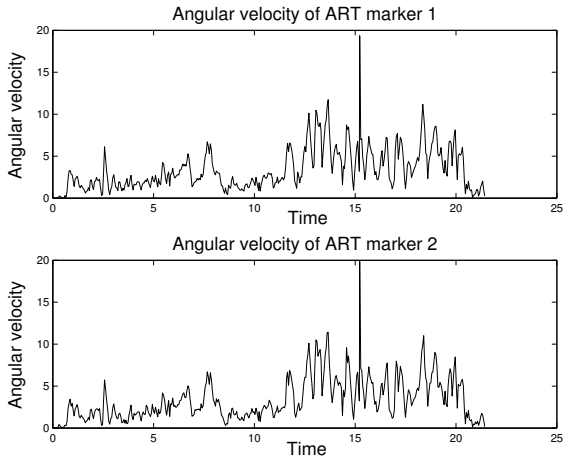



**Figure 7.10:** Angular velocity diagrams

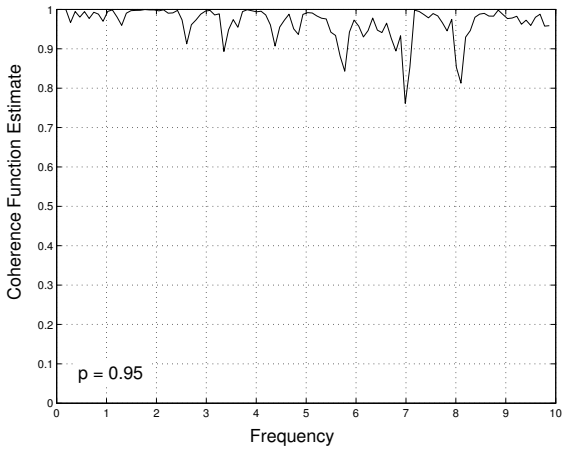



**Figure 7.11:** Coherence function of the angular velocity

Results of test case 3: Speed

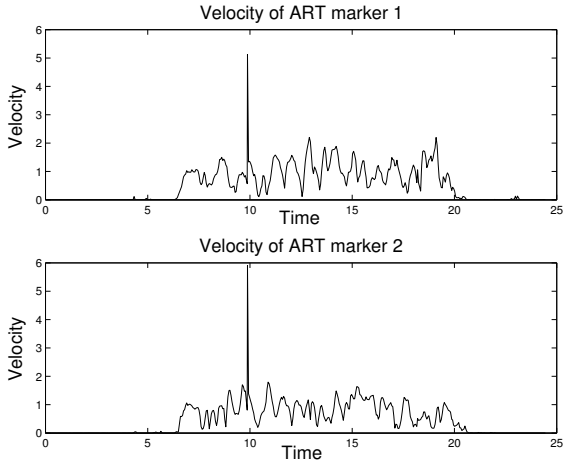



**Figure 7.12:** Speed diagrams

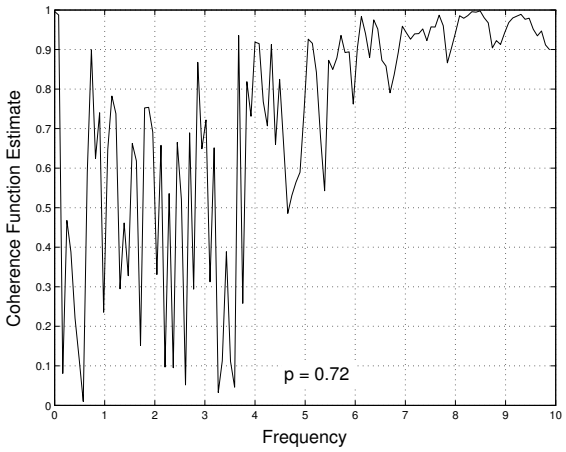



**Figure 7.13:** Coherence function of the speed

Results of test case 3: Angular velocity

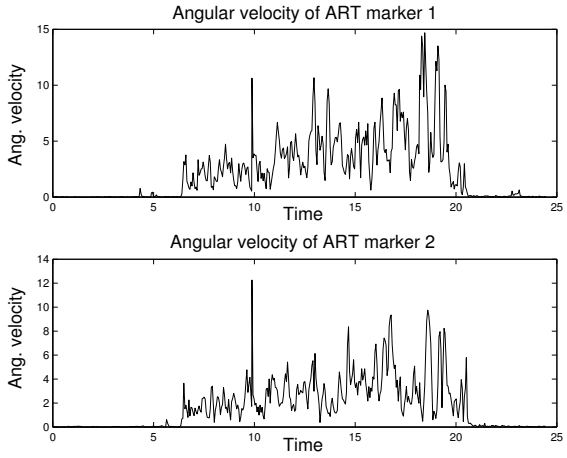



**Figure 7.14:** Angular velocity diagrams

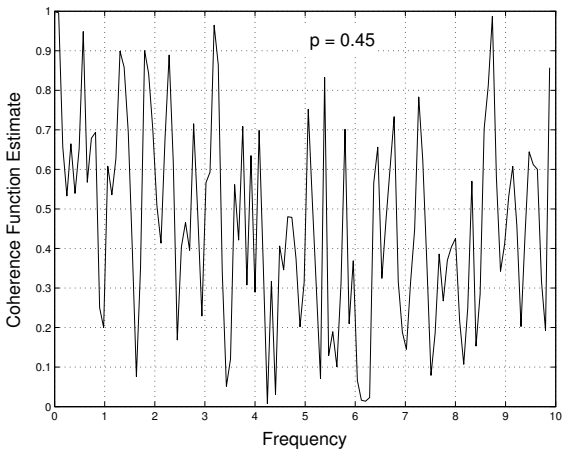



**Figure 7.15:** Coherence function of the angular velocity

Results of test case 4: Speed

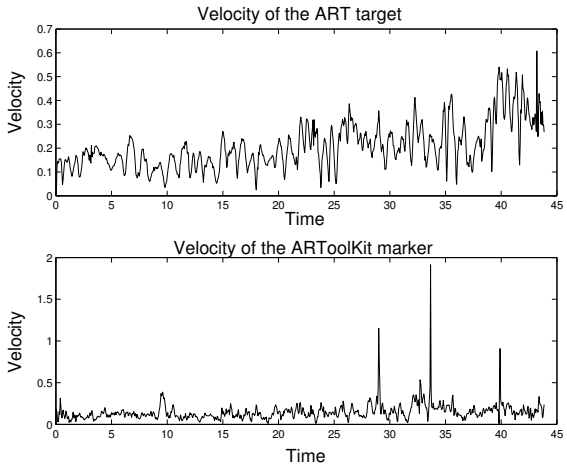



**Figure 7.16:** Speed diagrams

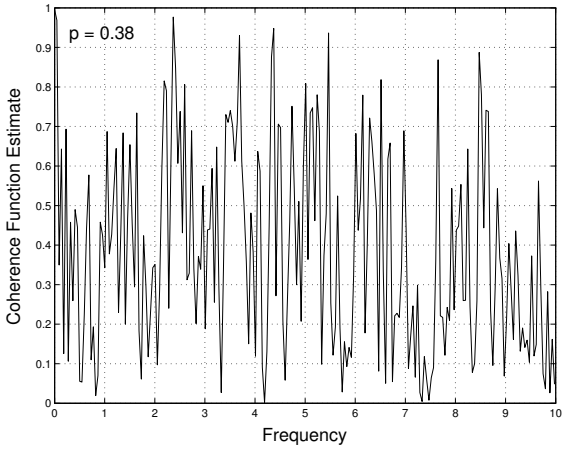



**Figure 7.17:** Coherence function of the speed

Results of test case 4: Angular velocity

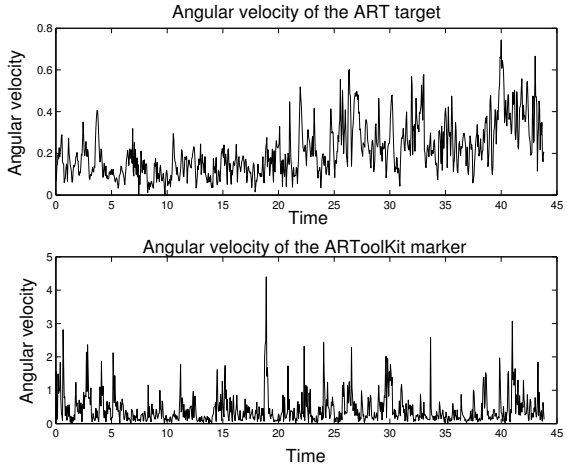



**Figure 7.18:** Angular velocity diagrams

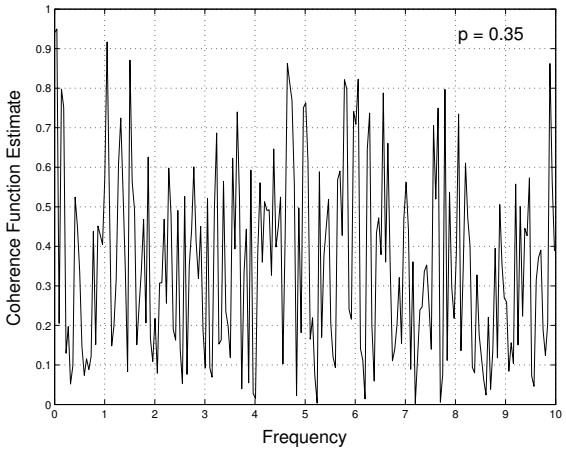



**Figure 7.19:** Coherence function of the angular velocity

Results of test case 5: Angular velocity

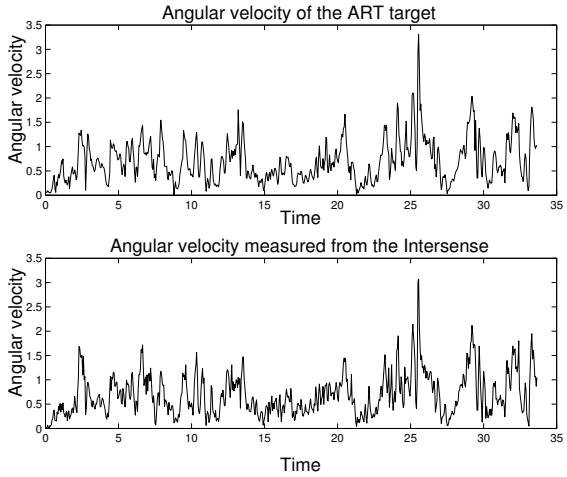



**Figure 7.20:** Angular velocity diagrams

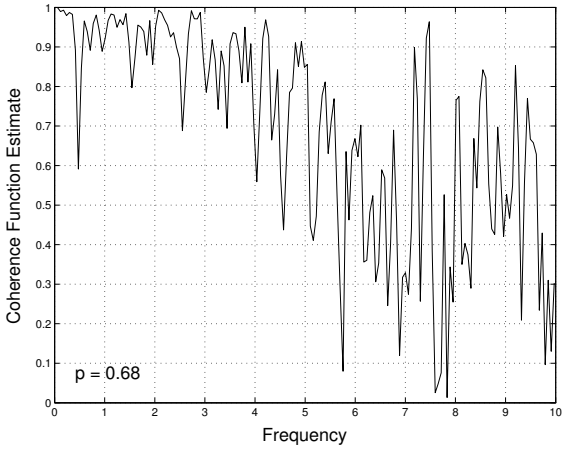



**Figure 7.21:** Coherence function of the angular velocity

# 8 Conclusion

**Lessons I learned during the thesis and future work.**

---

This chapter concludes my thesis. Firstly, I will present what I learned during my work with Ubitrack. Secondly, I will summarize all issues which are not covered in total by this thesis.

## 8.1 Lessons Learned

While writing this thesis I had to deal with problems which I did not work on in detail before; "tracking". Therefore I had to learn the theories behind "tracking" and came up with following results:

**Mathematical Basics for Tracking** The mathematical theories were difficult to understand at first glance. Especially when dealing with linear equations or quaternions for transformations, it seemed hard at the beginning. However especially in Augmented Reality, mathematics can be visualized intuitively. You just have to experiment with trackers and you will see the results of your work immediately — the false results as well as the correct ones.

**Frequency and Spectral Analysis** I had to learn these parts of mathematical analysis in no time, because the paper with the approach I focused on [34] was published in April 2004 and therefore it is state of the art. A good Fourier analysis depends on a appropriate windowing of the input signal. I used a similar windowing as proposed in the paper, but more intelligent window functions should be evaluated. Tuning the parameter of the window function and of the FFT resulted in great differences in the coherence.

**Adaption of the Frequency-Based Approach** I adapted the approach from acceleration to velocity without knowing whether our tracking systems provide sensible data or not. But they did. Sometimes. The most important problem was the data delivered by ARToolKit. At first I assumed that the linear interpolation of the `Inference` component adulterated the first data series. So I used the more advanced Kalman filter for a better interpolation. Whichever attempt, the results were equally poor. By analyzing the direct positing components, the diagrams showed that the ARToolKit signal delayed by 5 seconds compared to the original signal (given that the measurements were not completely wrong).

## 8.2 Future Work

Ubitrack was a project which demanded a lot of work, because of the complex theory in the whole tracking area. Understanding the formal model of Ubiquitous Tracking is the easy part, but finding a suitable implementation with all the dynamic concepts is difficult. Therefore many details of the arisen problems were difficult and could not be solved completely. The following list shows a summary of all problems which are still unsolved. This is followed by a list of steps, which can be undertaken afterward.

### 8.2.1 Open Issues in the Ubitrack Design

The current overall Ubitrack design tries to address all given requirements. But there is room for improvement:

**Analyzing Scalability Problems** Because Ubitrack should be a distributed and decentralized system, the application of any central components are critical. However, scalability means that the whole system is divided into smaller pieces which can manage themselves and therefore can be build into more complex systems without much additional overhead which would otherwise result in bottlenecks. Finding these bottlenecks in the Ubitrack implementation is crucial for the large-scale application of sensor networks. If our scalability considerations are covering everything is not guaranteed. It must be proven in large-scale experiments or simulations.

**Mobile Clients with More UMAs** The implementation design of the mobile scenarios assumed that the mobile client consists only of one UMA. This is not given in general. The general assumption would be that two UMA overlay networks connect to each other. This emerges the problem of how the location context should be propagated: Should every UMA of the mobile client be connected to the same ContextUMA? Or should there be a central UMA for every location context (so called SuperUMA) and should they connect? How are SuperUMAs determined at runtime? Are they determined by leader elections? What happens when they fail?

### 8.2.2 Open Issues in the Mobile Scenarios

As mentioned in chapter 4, there are open questions left when dealing with incomplete Spatial Relationship Graph with anonymous objects. When two sensors track two hitherto unknown objects they generate two anonymous objects. If they are already registered with their real IDs, there are four object nodes in the graph but in reality there are only two objects. Figure 4.10 on page 54 shows this situation. Four possible mappings for real object identifiers for the anonymous objects exist:

1. $X = M$ and $Y = N$
2. $X = N$ and $Y = M$
3. $X = M$ and $Y = M$
4. $X = N$ and $Y = N$

This case has to be investigated in the future and has to be integrated in the formal description in chapter 4.

### 8.2.3 Open Issues in the Dwarf Implementation

Some issues are not just inefficiently discussed in terms of design and scenarios, but also how those can be implemented properly.

**Dwarf Solves and Creates Problems** Working with DWARF solved the problem of a decentralized approach but introduced possible bottlenecks resulting from not scalable mechanisms of the Service Manager. Especially the *location subsystem* of the DWARF Service Manager is critical for being used in Ubitrack. In terms of the Service Manager, this is the location phase of services. The current implementation of the manager relies on SLP broadcasts in order to detect services. Especially when considering thousands of sensors with their hardware description services, this will not scale to an arbitrary extend. Therefore the introduction of the UMA was necessary.

**Delegation of Service Manager Functionality to the UMA** In general the UMA was designed to provide possible replacement strategies for parts of the manager which meet the requirements of Ubitrack in a better way. Since it already manages the location context which divides the network in pieces according to their spatial neighborhood, it has all relevant information about close-by objects. Why should not the location phase for hardware descriptions delegated to the UMA?

**Migrate the UMA into the Middleware** The original design of the DWARF middleware [36] was not limited to the Service Manager. The middleware can consist of more *Distributed Mediating Agents*. The UMA would perfectly fit into this scheme.

**Further Issues** The following list shows which features are currently missing and have to be tested in an actual full implemented demo set-up:

- **Hierarchical context attributes**:
  It must be evaluated if hierarchical context attributes can be implemented by the normal DWARF attribute and predicate scheme.
- **Hardware description services**:
  Actually, they only make sense in large-scale sensor networks. If it is suitable for thousands of sensors and locatables or if it can be used for *natural feature-based tracking* at all, must also be evaluated.
- **Scalability**:
  The proposed location context model allows a scalable approach for the exchange of the hardware descriptions. However it is realized by Service Manager means completely. Performance measurements must prove whether it is sufficient to rely on the current DWARF implementation or not.

### 8.2.4 Open Issues when Estimating Spatial Relationships

The theoretical chapter about estimation relationships without actual measurements brings up most of the questions.

**More Efficient and Hybrid Approaches** Because the estimations must present results as fast as possible computational expensive approaches must be avoided. However, fast approaches could not be accurate enough for a sensible result. So probably the successive application of different methods are to chose. Starting with in inaccurate fast algorithm which eliminates relationships which are clearly unstatic. For ambiguous cases more precise methods can be applied.

**Weighting the Coherence** For improving the probability values it is reasonable to weigh low frequencies more than higher ones. This is due to the fact that measured noise results more likely in a tremble at higher frequency bands. For that purpose, a suitable probability density function could replace the assumed normal distribution in equation 6.11 on page 72. The used probability function for the equal distribution in the test cases, $d(f) = \frac{1}{10}$, is certainly suitable to compare motion patterns. Starting from the general equation for the probability value

$$p = \int_0^{10} d(f) C_{xy}(f)\, df \tag{8.1}$$

we can evaluate several density functions $d(f)$ with

$$\int_0^{10} d(f)\, df = 1. \tag{8.2}$$

For example we can use a density function similar to a normal distribution, a falling exponential function:

$$d(f) = ae^{-bf^2} \tag{8.3}$$

The parameters $a$ and $b$ must fulfill the density criteria in equation 8.2 and must model the weighting in a realistic way.

### 8.2.5 Next Steps

After the list of problems which are still unsolved, here are some steps which could be realized in the near future.

**Implementing and Integrating the Concepts** Starting from this thesis and the available components it should be possible to implement a first dynamic set-up.

**Simulating a Large-Scale Demo Set-up** Because of the absence of hardware, a large-scale demo set-up must be simulated. Therefore a suitable simulator can be used, for example the already existing simulator for the Ubitrack implementation at the Vienna University of Technology [54].

# A Glossary

This chapter lists important terms and abbrevations which are essential for a basic understanding of this thesis.

**Ability** If a DWARF → *service* has an ability, it provides certain information for other services. Every ability has a type which specifies what kind of data is provided exactly. Abilities can have → *attributes* which describe the data by additional information.

**Attribute** Attributes are name-value-pairs which specify additional information for → *abilities*. The attribute's name is donated by `name=value` where the value can be omitted. Attributes are modeled by circles in UML model diagrams.

**Coherence** The coherence is a function which computes a similarity function of two → *signal* in the frequency domain. It is the Fourier transformed function of the correlation function.

**DFT** The Discrete Fourier Transform (DFT) is the transformation function which maps discrete time-dependent functions into the frequeny domain.

**DWARF** DWARF is the Distributed Wearable Augmented Reality Framework. It provides → *services* as components for rapid-prototyping AR applications. It uses a distributes Ubicomp architecture based on a middleware, called → *Service Manager*.

**FFT** The Fast Fourier Transform is an efficient way to compute the Discrete Fourier Transfrom (→ *DFT*) for discrete time steps.

**Hardware Description Service** Such service provides configuration information about a hardware component for which it is designed. Currently, → *sensors* and → *locatable* may provide description services. In Ubitrack they are represented as → *object nodes* in the → *Spatial Relationship Graph*.

**Locatables** Locatables are physical characteristics which are trackable by → *sensors*. These characteristics include e.g. visual properties of artifical things, sound and even natural features.

**Measurement edges** These directed edges in the → *Spatial Relationship Graph* models spatial relationships of → *object nodes*. The relationship describes the location and/or orientation of the object of the in-going node in respect to the frame of reference of the out-going node. Measurment edges are realized in Ubitrack by → *tracking services*

**Need** If a DWARF → *service* has a desire for certain data, it provides a need. Needs can specify → *predicates* which may restrict the matching → *abilities*: only if the need's predicate is matching the attributes, the two services gets connected.

**Object nodes** Nodes in the → *Spatial Relationship Graph* which represents real world objects. Every object is specified by an identifier which is unique throughout the graph. Object nodes are realized in Ubitrack by → *hardware description services*. → *Sensors* and → *locatables* are examples for these nodes.

**PoseData** PoseData is a) the name of an → *ability* or → *need* to send or receive spatial information or b) the name of the data structure containing the spatial information, normally sent by events.

**Power Spectrum** The power spectrum shows the energy at every frequency value. This indicates the intensity of the frequency component in the original → *signal*.

**Predicate** → *Needs* can have predicates to restrict matching → *abilities*. Only if the predicate matches the attributes, two → *services* get connected to each other by the DWARF Service Manager. Predicates are drawn as semi-circles in UML model diagrams.

**Sensors** In Ubicomp, the term sensor refers to all kind of hardware devices which measure physical characteristics. For Ubitrack the definition is restricted to devices which deliver spatial information. The type of information which is provided ranges from classical spatial information of advanced → *trackers* (such as postition and orientation) to simple information of e.g. photoelectric barriers which can be used to improve the spatial relationships. Sensors are represented by → *object nodes* in the → *Spatial Relationship Graph*.

**Service** A service is a component in the DWARF application. It can provide its features by → *abilities* or request data by → *needs*.

**Service Manager** This is a part of the middleware of DWARF. It manages a repository of DWARF Service Descriptions for every local service. It is responsible for locating and connecting → *services* in ad-hoc networks.

**Signal** A signal is any data set which is dependent on time. In this thesis, a typical signal is the velocity.

**Spatial Relationship Graph** A graph containing the spatial relationships of objects in the real world. It consists of → *object nodes* and → *measurement edges*.

**Tracker** Trackers are advanced → *sensors* which deliver classical spatial information, such as position and orientation.

**Tracking Services** Tracking services are DWARF → *services* which consists of → `PoseData` abilities to deliver spatial information within DWARF applications. In Ubitrack they are represented as → *measurement edges* in the → *Spatial Relationship Graph*.

**UMA** This is the Ubitrack Middleware Agent. It is responsible for the graph search and for managing the location context. One UMA resides on every network node and manages the local → *sensors* and → *locatables*.

# Bibliography


[1] Danette Allen, Gary Bishup, and Greg Welch. Tracking: Beyond 15 minutes of thought. Chapel Hill, NC 27599-3175, 2001. Course Notes SIGGRAPH. 1.4, 1.4.2

[2] ARCHIE project homepage. `http://wwwbruegge.in.tum.de/DWARF/ProjectArchie`. 1.2, 3.2.2, 4.2, 5.4.2

[3] R. Azuma. A survey of augmented reality, 1995. 1.2, 1.4.2

[4] R. Azuma and G. Bishop. Improving static and dynamic registration in an optical see-through hmd. In *Proc. Siggraph'94*, pages 194–204, Orlando, July 1994. 1.5.2, 2.1.3, 2.5

[5] Ronald Azuma, Bruce Hoff, Howard Neely, and Ron Sarfaty. A motion-stabilized outdoor augmented reality system. In *VR*, pages 252–259, 1999. 1.2

[6] Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Stefan Riss, Christian Sandor, and Martin Wagner. Design of a component-based augmented reality framework. In *Proceedings of the 2nd IEEE and ACM International Symposium on Augmented Reality (ISAR 2001)*, New York, NY, October 2001. IEEE Computer Society. 2.2

[7] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 1990. 4.3.2

[8] T. Bernes-Lee, R. Fielding, U. C. Irvine, and L. Masinter. RFC 2396: Uniform resource identifiers (URI): Generic syntax, August 1998. Updates: RFC 1808, RFC 1738. 3.5.2

[9] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992. 6.3

[10] Dagmar Beyer. Construction of decentralized data flow graphs in ubiquitous tracking environments. Master's thesis, Technische Universität München, 2004. 1.5, 1.6, 2, 4.4.2, 5.3.2

[11] Ilja N. Bronstein, Konstantin A. Semendjajew, and Gerhard Musiol. *Taschenbuch der Mathematik*. Harri Deutsch, Frankfurt a.M., 5th edition, August 2000. 6.6.1

[12] Barry Brumitt, John Krumm, Brian Meyers, and Steven Shafer. Ubiquitous computing & the role of geometry. *IEEE Personal Communications*, October 2000. 1.3, 3.5.4

[13] C. Carter. Tutorial overview of coherence and time delay estimation. In *Coherence and Time Delay Estimation – An Applied Tutorial for Research, Development, Test, and Evaluation Engineers*, pages 1–23. IEEE Press, 1993. 6.6.1

[14] Enylton Machado Coelho and Blair MacIntyre. High-level tracker abstractions for augmented reality system design. In *International Workshop on Software Technology for Augmented Reality Systems*, pages 12–15, October 2003. 2.1.2

[15] George Coulouris, Hani Naguib, and Kam Samugalingam. FLAME: An open framework for location-aware systems. Technical report, University of Cambridge, Lab for Communication Engineering, 2002. `http://www-lce.eng.cam.ac.uk/qosdream/Publications/flame.pdf`. 1.3.3

[16] Murat Demirbas and Hakan Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Third International Conference on Peer-to-Peer Computing*, pages 32–39, 2003. 4.3.2, 5.3.1

[17] Anind K. Dey and Gregory D. Abowd. The context toolkit: Aiding the development of context-aware applications. Technical report, Georgia Institute of Technology, Ubiquitous Computing Research Group, Atlanta, USA, 2000. `http://www.cc.gatech.edu/fce/contexttoolkit`. 1.3.3

[18] R. Droms. RFC 2131: Dynamic host configuration protocol, March 1997. Obsoletes RFC1541. Status: DRAFT STANDARD. 3.2.1

[19] DWARF project homepage. `http://www.augmentedreality.de`. 2.2

[20] Martin Lillholm Erik B. Dam, Martin Koch. Quaternions, interpolation and animation. Technical report, Department of Computer Science, University of Copenhagen, 1998. 1.4.1, 2.4.1, 6.4.2

[21] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks, 2001. 1.3.3

[22] Eric Foxlin. Inertial head-tracker sensor fusion by a complimentary separate-bias kalman filter. In *Proceedings fo the 1996 Virtual Reality Annual International Symposium (VRAIS 96)*, page 185, March 1996. 2.5

[23] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984. 4.3.2

[24] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol (SLP), version 2, June 1999. Updates: RFC 2165. 3.5.2, 5.4.2

[25] Drexel Hallaway, Tobias Höllerer, and Steven Feiner. Bridging the gaps: Hybrid tracking for adaptive mobile augmented reality. *Applied Artificial Intelligence, Special Edition on Artificial Intelligence in Mobile Systems*, 25(5), July 2004. 2.5

[26] Wendi Beth Heinzelman. *Application-Specific Protocol Architectures for Wireless Networks*. PhD thesis, Massachusetts Institute of Technology, June 2000. 1.3.3

[27] Björn Hermans. Desperately seeking: Helping hands and human touch. Published online at `http://www.hermans.org/agents2/`, 1998. 1.3.1

[28] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000. 1.3.3

[29] William Hoff and Tyrone Vincent. Analysis of head pose accuracy in augmented reality. *IEEE Transactions on Visualization and Computer Graphics*, 6(4):319–334, October–December 2000. 2.5

[30] Tobias Höllerer, Drexel Hallaway, Navdeep Tinna, and Steven Feiner. Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system. In *Second Int. Workshop on Artificial Intelligence in Mobile Systems*, page 31, August 2001. 2.1.2, 2.5

[31] Intel research – silicon – moore's law. `http://www.intel.com/research/silicon/mooreslaw.htm`. 1.1

[32] H. Kato and Mark Billinghurst. Marker tracking and HMD calibration for a video-based augmented reality conferenencing system. In *Proc. IWAR'99*, pages 85–94, San Francisco, CA, USA, October 21–22 1999. IEEE CS. 1.4.2, 2.3

[33] Hirokazu Kato, Mark Billinghurst, and Ivan Poupyrev. *ARToolKit version 2.33 Manual*, 2000. Available for download at `http://www.hitl.washington.edu/research/shared_space/download/`. 1.4.2

[34] Jonathan Lester, Blake Hannaford, and Gaetano Borriello. "Are You with Me?" - using accelerometers to determine if two devices are carried by the same person. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing: Second International Conference, PERVASIVE 2004*, number 3001 in LNCS, pages 33–50, Linz/Vienna, Austria, April 2004. Springer-Verlag. 6.6, 6.6.1, 6.6.1, 6.6.1, 6.6.2, 6.6.4, 8.1

[35] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc, 1996. 2.1.2, 2.4.4

[36] Asa MacWilliams. Using ad-hoc services for mobile augmented reality systems. Master's thesis, Technische Universität München, 2001. 8.2.3

[37] Asa MacWilliams, Thomas Reicher, and Bernd Brügge. Decentralized coordination of distributed interdependent services. In *IEEE Distributed Systems Online – Middleware Work in Progress Papers*, Rio de Janeiro, Brazil, June 2003. 5.3.1

[38] Matlab online help. Reference for MATLAB function fft. 6.6.1

[39] Gordon Matzigkeit and Yoshinori K. Okuji. *the GRUB manual*, 2004. 3.2.1

[40] P. Milgram and H. Colquhoun. *A taxonomy of real and virtual world display integration*, chapter 1, pages 5–30. Ohmsha (Tokyo) and Springer Verlag (Berlin), 1999. 1.1, 1.2

[41] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, Volume 38(8), April 1965. 1.1

[42] Joseph Newman, Martin Wagner, Thomas Pintaric, Asa MacWilliams, Martin Bauer, Gudrun Klinker, and Dieter Schmalstieg. Fundamentals of ubiquitous tracking for augmented reality. Technical Report TUM-I0323, Technische Universität München, December 2003. 1.6

[43] Donald A. Norman. *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution*. MIT press, reprint edition, August 1999. 1.1.1

[44] Nyquist-Shannon sampling theorem. Wikipedia article at `http://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem`. 6.6.1

[45] Object Management Group. *Common Object Request Broker Architecture (CORBA/IIOP)*, March 2004. current revision 3.0.3. 2.2.2

[46] OSD: PC bootstrap. `http://my.execpc.com/~geezer/osd/boot/`. 3.2.1

[47] Charles E. Perkins and Elizabeth M. Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, February 1999. 3.2.1

[48] Daniel Pustka. Autocalibration in ubiquitous tracking environments. Master's thesis, Technische Universität München, 2004. 1.5, 1.6, 2, 2.1.2, 2.4.5, 4.4.2, 7.5

[49] Gerhard Reitmayr and Dieter Schmalstieg. Opentracker - an open software architecture for reconfigurable tracking based on xml. In *Proceedings of the Virtual Reality 2001 Conference (VR'01)*, page 285, March 2001. 2.5

[50] Apple developer connection: Rendevous services. `http://developer.apple.com/documentation/Cocoa/Conceptual/NetServices/index.html`. 3.2.1

[51] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings of the Third International Conference on 3-D Digital Imaging and Modeling (3DIM)*, pages 145–152, Quebec City, Canada, May 2001. 6.3.2

[52] Christian Sandor, Asa MacWilliams, Martin Wagner, Martin Bauer, and Gudrun Klinker. SHEEP: The shared environment entertainment pasture. In *IEEE and ACM International Symposium on Mixed and Augmented Reality ISMAR 2002*, 2002. 1.2, 5.4.2

[53] Franz Strasser. Personalized ubiquitous computing with handhelds in an ad-hoc service environment. Systementwicklungsprojekt, Technische Universität München, 2003. 3.1, 3.2.2, 5.3.1

[54] The studierstube project homepage. `http://www.studierstube.org/`. 8.2.5

[55] Russell M. Taylor, ll, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. Vrpn: a device-independent, network-transparent vr peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 55–61. ACM Press, 2001. 2.5

[56] Marcus Tönnis. Data management for AR applications. Master's thesis, Technische Universität München, 2003. 5.4.1

[57] Roel Vertegaal. Introduction: Attentive user interfaces. *Commun. ACM*, 46(3):30–33, 2003. 1.3

[58] Martin Wagner and Gudrun Klinker. An architecture for distributed spatial configuration of context aware applications. In *2nd International Conference on Mobile and Ubiquitous Multimedia*, Norrköping, Sweden, 2003. 4.2, 5.4.3

[59] Mark Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, September 1991. 1.3

[60] Eric W. Weisstein. Power spectrum. From Mathworld–A Wolfram Web Resource `http://mathworld.wolfram.com/PowerSpectrum.html`. 6.6.1

[61] Gregory Francis Welch. *SCAAT: incremental tracking with incomplete information*. PhD thesis, University of North Carolina at Chapel Hill, 1996. 2.1.3

[62] A. Williams. Requirements for automatic configuration of ip hosts, September 2002. Internet Draft draft-ietf-zeroconf-reqts-12.txt. 3.2.1

[63] David A. Winter. *Biomechanics and Motor Control of Human Movement*. Wiley, 2nd edition, 1990. 6.6.1

[64] Suya You and Ulrich Neumann. Fusion of vision and gyro tracking for robust augmented reality registration. In *Proceedings of the Virtual Reality 2001 Conference (VR'01)*, page 71. IEEE Computer Society, 2001. 2.1.3

[65] Suya You, Ulrich Neumann, and Ronald Azuma. Hybrid inertial and vision tracking for augmented reality registration. In *Proceedings of the IEEE Virtual Reality*, page 260. IEEE Computer Society, 1999. 2.5

[66] IETF Zeroconf homepage. `http://www.zeroconf.org/`. 3.2.1