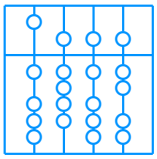
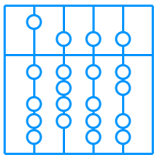
Institut für Informatik der
Technischen Universität München



Tracking with Multiple Sensors

Martin Wagner

Institut für Informatik der
Technischen Universität München



Tracking with Multiple Sensors

Martin Wagner

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Dr. h.c. Wilfried Brauer

Prüfer der Dissertation:

1. Univ.-Prof. Gudrun J. Klinker, Ph.D.
2. Univ.-Prof. Dr. Dieter Schmalstieg,
   Technische Universität Graz/Österreich

Die Dissertation wurde am 9.12.2004 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11.03.2005 angenommen.

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der dynamischen Kombination mehrerer mobiler und stationärer Sensoren zur Ortsbestimmung in intelligenten Umgebungen.

Durch eine Analyse verwandter Arbeiten in den Bereichen der Augmented Reality (AR) und des Ubiquitous Computing (Ubicomp) werden stark unterschiedliche Anforderungen dieser Forschungsgebiete an das Aufgabengebiet der Arbeit definiert: typische Anwendungen der AR benutzen wenige hochgenaue Sensoren, die Ortsdaten mit niedriger Latenzzeit und hoher Wiederholrate liefern. Anwendungen aus dem Gebiet des Ubicomp benutzen hingegen meist eine große Anzahl von Sensoren, die recht ungenaue Daten mit höchst unterschiedlichen Wiederholraten liefern, die zudem nur einen Ausschnitt des Ortszustandes der beobachteten Objekte darstellen. Eine einheitliche Behandlung von Sensoren zur Ortsbestimmung in den Gebieten von AR und Ubicomp ist eine zwingende Voraussetzung für AR-Benutzerschnittstellen in Ubicomp-Umgebungen, allerdings wurde dies in bisherigen Arbeiten vernachlässigt. Zudem behandelten existierende Mehrsensorsysteme in den Bereichen der AR und des Ubicomp kaum das Problem der dynamischen Integration von mobilen Benutzern und ihrer mitgeführten Sensoren und Anwendungen zur Laufzeit. Die vorliegende Arbeit legt Lösungsansätze für diesen im folgenden *Ubiquitous Tracking (Ubitrack)* genannten Problembereich dar.

Der vorgestellte graphbasierte Formalismus erlaubt die einheitliche Behandlung aller relevanter Sensornetze zur Ortsbestimmung. Er wird in den folgenden Teilen der Arbeit sowohl als grundlegendes theoretisches Modell als auch zur exakten und vereinfachten Beschreibung von Sensornetzen benutzt. Um den Formalismus zur Lösung von Anwendungsproblemen nutzen zu können, wird ein Implementierungskonzept beschrieben, das auf verteilten Algorithmen zur Sensordatenerfassung und -aggregation beruht. Die Realisierbarkeit sowohl des Formalismus als auch des Implementierungskonzepts wird durch Simulation der zugrundeliegenden Algorithmen gezeigt.

Der Entwurf und die Implementierung einer Softwarearchitektur zur dynamischen Integration neu auftauchender mobiler Anwendungen und Sensornetze in existierende stationäre Aufbauten wird beschrieben. Die Architektur basiert auf früheren Arbeiten am Augmented Reality Framework DWARF. Dabei wird ein besonderer Schwerpunkt auf Lösungsansätze zur Übertragung von Konfigurationsdaten in Peer-to-Peer Netzen gelegt, so dass a priori weder die mobilen noch die stationären Systeme voneinander Informationen haben müssen.

Zum Abschluss wird die verteilte Anwendung *Ubiquitous SHEEP* vorgestellt. Sie zeigt die wesentlichen Lösungsansätze der vorliegenden Arbeit im praktischen Zusammenspiel.

Abstract

This thesis aims to find new ways of dynamically combining multiple mobile and stationary sensors for location tracking in intelligent environments.

Based on an overview of existing location sensors and sensor fusion systems both in the area of Augmented Reality (AR) and Ubiquitous Computing (ubicomp), very different requirements for these two fields of research are derived: AR applications typically employ positional data of a few very accurate high-frequency, low-latency sensors with six degrees of freedom, whereas ubicomp systems use a large number of sensors that usually are much less accurate, deliver data with highly variable update rates and give only partial information on a tracked object's position and orientation. In consequence, there has not been much work on a unified treatment of location sensors for AR and ubicomp, which is a prerequisite for interacting with ubicomp environments via AR-based interfaces. In addition, existing multi-sensor AR and ubicomp systems do not focus on the dynamic integration of new mobile sensors, users and applications at runtime. However, in intelligent environments mobile users carrying their own equipment have to be connected with stationary systems and among each other at runtime to exploit all available data and enable new kinds of applications. This thesis is a first step in this direction and gives partial solutions to this problem domain, which is referred to as *Ubiquitous Tracking (Ubitrack)* throughout the thesis.

A graph-based formalism is presented that allows the unified treatment of all kinds of sensor networks for location tracking. The formalism is used both as an underlying theoretical model and a way of describing sensor networks in the remainder of the thesis. The design of an implementation concept consisting of distributed algorithms extracting and aggregating location sensor data to apply the formalism to real-world problems is described. In contrast to existing approaches, it allows the dynamic integration of mobile setups at runtime. To prove the viability of both the formalism and the implementation concept, a simulation of the underlying algorithms is presented.

Based on previous work on the DWARF Augmented Reality framework, the design and implementation of a software architecture that allows the dynamic integration of new mobile sensor networks and applications into existing stationary setups is discussed. A major issue is how to transmit configuration data in a peer-to-peer fashion, such that neither mobile nor stationary setups must have a priori knowledge about each other on system startup.

As a proof of concept, a game-playing application, *Ubiquitous SHEEP*, is described, demonstrating all key features described in this thesis in practice.

Acknowledgments

Writing a thesis is, in principle, a process of endless pain and suffering. Nevertheless, many people helped in making the last three years a joyful and extremely interesting experience.

First and foremost I thank my advisor Prof. Gudrun Klinker, Ph.D., for calm and prudent advice on how to do research, how to present my own findings and how to express the relevance of new topics. A maximum of freedom in both what I did and how I did it was accompanied by constant care if any problems arose. I thank Prof. Dr. Dieter Schmalstieg for fruitful discussions and for having had the opportunity to stay within his research group in January 2004. I thank Prof. Bernd Brügge, Ph.D., for providing the opportunity to work at his chair.

I am thankful to Joe Newman for detailed discussions—not only on Ubitrack and computer science issues—many happy days in the Vienna bike room, and an ideal collaborative atmosphere. Thanks to all participants of the Ubitrack project for insightful workshops, providing valuable input and refining concepts. I am much obliged to my colleagues from the Augmented Reality Research Group at Technische Universität München: Martin Bauer, Verena Broy, Stephan Huber, Asa MacWilliams, Hesam Najafi, Thomas Reicher, Christian Sandor and Marcus Tönnis offered both an excellent academic environment and friendship. The degree of collaboration and support within this group is hard to find anywhere else.

A great thank you to all students I had the opportunity to work with in term or diploma theses: Gerrit Hillebrand, Marcus Tönnis, Bernhard Zaun, Felix Löw, Andreas Krause, Florian Reitmeir, Günther Harrasser, Sven Hennauer, Fuat Atabey, Recep Kayali, Franz Mader, Daniel Pustka, Dagmar Beyer, Jonas von Beck, Ingo Kresse, Christian Wachinger and Michael Siggelkow. Special thanks go to Franz Strasser, who did a tremendous job in developing crucial parts of the Ubiquitous SHEEP demo. Working with you was a pleasure.

This work would not have been possible without the enduring support and motivation of my family who unconditionally supported my interests in computer science at all times. I am also indebted to my friends for keeping my feet on the ground and many joyful and relaxing evenings. In particular, I would like to thank Maria Bauer, Stefan Beyer, Denise Chlosta, Tina Dangl, Thomas Purrucker, Christian Sandor, Marcus Tönnis, Alexander Wotzko and Bernhard Zaun. Special thanks to Kati List for extraordinary patience, especially within the last weeks of writing.

I thank all reviewers of draft versions of this thesis: Gudrun Klinker, Dieter Schmalstieg, Albrecht Schmidt, Mihran Tuceryan, Martin Bauer, Dagmar Beyer and Franz Strasser.

This thesis was supported by the High-Tech Offensive der Bayerischen Staatskanzlei (Projekt UMTS–Mobile Wartung) and the Deutsche Forschungsgemeinschaft (Projekt DySenNetz, KL1460/1-1).

Contents

CHAPTER 1

Introduction

Overview

This thesis aims at finding new ways of dynamically combining multiple sensors for location tracking in intelligent environments.

This chapter introduces and defines the two main areas of research this thesis relates to. The first area, Augmented Reality (AR), is a new paradigm for interfacing with computing systems. The influence of the properties of typical AR systems on the requirements for sensors that estimate the positions and orientations of users or objects in the users' environment is described.

Especially in terms of location tracking, research in AR has been inspired by ideas of Virtual Reality (VR), where the user is taken to a complete artificial world. The second area, Ubiquitous Computing (ubicomp), forms a sharp contrast to that approach. Instead of taking the user to virtual worlds, it aims at bringing virtual information to the user, as Weiser [131] puts it:

> The "virtuality" of computer-readable data—all the different ways in which they can be altered, processed and analyzed—is brought into the physical world.

I argue that user interface paradigms of Augmented Reality are—among others—a perfect match for interfacing with ubicomp environments. Only by combining virtuality and reality in three dimensions it is possible to reach the ubicomp goal of overcoming the information overload omnipresent in today's desktop based computer systems. To motivate this claim, some scenarios illustrating the potential benefits of AR interfaces in ubicomp environments are given.

At the current state of research, the sensing technology of ubicomp applications is largely different from AR's technology: research in ubicomp focusses on flexibility–ad-hoc connections of computers providing sensor information and getting contextual information out of omnipresent sensors are main areas of attention. In contrast, AR researchers focus on sensors delivering high positional and orientational accuracy. The work described in this thesis is a first step at reaching these presumably contradictory goals simultaneously.

1.1 Augmented Reality

Being inspired by the vision of taking the information within a computing system to the user's perception of the real world, the idea of Augmented Reality (AR) has been invented by Ivan E. Sutherland [116] shortly after his pioneering work on Virtual Reality (VR) [115]. Whereas VR systems aim at creating seemingly realistic, but completely artificial worlds, AR systems go one step further and try to mix artificial and realistic impressions such that the user is supported in performing real-world tasks rather than in manipulating merely virtual data. This section gives a definition of AR, and briefly presents the current state of the art in the research community. As this thesis is all about tracking, special focus is put on a presentation of today's AR sensing technology and its main characteristics.

1.1.1 Definition

Azuma summarized the AR development in two survey papers [7, 8]. Out of the vast literature on existing AR systems, he distilled a definition that nowadays is commonly accepted among AR researchers.

An Augmented Reality system

1. combines real and virtual objects in a real environment;
2. runs interactively, and in real time; and
3. registers (aligns) real and virtual objects with each other in three dimensions.

There are several interesting observations to make about this definition. Many people's idea of AR systems is restricted to visual augmentations, either using Head-Mounted Displays (HMDs) or other visual techniques such as projectors displaying data on a table. In principle, these are valid AR systems, but it is equally fine to employ other senses of the users, e.g. by using sound or force feedback devices. A virtual object does not have to be seen by a user to make it present, it might suffice that the user hears where it is. As long as the virtual object is aligned in an interactive fashion according to some three-dimensional spatial relationship to a real counterpart, the overall system is an AR system.

The real time requirement makes AR difficult. Combining real and virtual objects that are registered in three dimensions is a rather standard technique in today's movie industry, both for visual and audio effects. It is often almost impossible to distinguish the real from the virtual parts. However, creating such scenes is an offline process that takes many days to render on high-performance machines. If the whole AR processing pipeline—detecting the pose of real objects in 3D space, aligning virtual objects with them and bringing the virtual objects to the user's perception of the real world by e.g. displaying them—has to work in "interactive real time", i.e. there are just a few milliseconds to perform all these computationally expensive tasks. Of course, increasing computer power facilitates this problem, but AR is still in a state where care has to be taken to implement every step in the processing pipeline as efficiently as possible.

Finally, the registration requirement takes us to the topic of this thesis. In AR applications, we have to *track* a multitude of real objects such that the virtual information can be aligned correctly. Conceptually, tracking is not limited to the 3D position and/or orientation of a rigid object in space, but should also handle deformable objects. We will see, however, that the current technology usually restricts us to estimate the position and orientation of a few rigid objects in 3D space.

1.1.2 State of the Art

Analyzing the proceedings of last years' *International Symposium on Mixed and Augmented Reality (ISMAR)* conference series [52, 53, 54, 55, 56], several major research directions can be observed:

- Researchers are still on the quest for the *killer application*, i.e. the application area that allows AR to become commercially successful (compare Navab [85]).

- A huge problem for developing real-world AR systems is finding suitable displays, as especially current head-mounted displays (HMDs), but also current projector-based techniques suffer from severe drawbacks. HMDs can either operate in video see-through mode, where a video image of a camera mounted in front of the display is augmented with virtual objects and shown in the user's goggles, or in optical see-through mode, where the real image of the world is optically merged with augmentations. Video see-through HMDs suffer from a low resolution of all available and processable cameras compared to the human eye, whilst optical see-through HMDs lack sufficient contrast to operate in varying lighting conditions. In addition, it is not possible to "take away" things from the user's view with optical see-through HMDs, a technique referred to as *diminished reality*. The same holds true for projectors, their advantage that the user does not have to wear special equipment

for visual augmentations is traded in for severe problems in multi-user setup, where it becomes extremely hard to ensure private information spaces with public displays.

- AR defines a completely new interaction paradigm with information systems. As AR is inherently three dimensional, immerses into the user's perception of his surroundings and AR systems also integrate real objects, knowledge from classic two dimensional user interfaces (UIs) can not be applied without major drawbacks. As such, much work is put in finding new UI metaphors suitable for AR systems. A promising approach is to use multi modal and tangible interfaces that take advantage of the fact that the user is in a real environment instead of the completely artificial world of conventional systems. However, it is still unclear which interaction techniques are suitable and commonly understandable for AR system users.

- Several groups, among them the Augmented Reality Group at Technische Universität München, which I am a member of, have put an emphasis on the system aspects of AR. Only if principles of software engineering are taken into account when developing AR applications will it be possible to reuse parts of AR systems and to build larger systems that can be integrated into existing business processes. The work described in this thesis tries to give a solution for the problem of reusing and combining tracking data in arbitrary AR systems.

- With the combination of real and virtual information, AR seems to be an ideal interfacing technology for mobile applications. However, there have been surprisingly few of them, which is mainly for missing tracking facilities—most outdoor applications are restricted to GPS—and insufficient computing power of mobile devices, which is currently becoming less of a problem. Still, most existing AR systems are restricted in their working area to a few square meters.

- The largest fraction of AR researchers is currently working on tracking technology. During the first years of AR, the major focus was on developing new technologies for tracking, and only some people started to eliminate the drawbacks of individual tracking technologies by *fusing* multiple tracking sources. This distribution holds true until today. Currently, many people work on vision-based tracking devices, with the major goal of using natural features such as sharp corners in structured environments as navigation hints. Vision-based natural feature tracking has the major advantage that no infrastructure is needed, a simple video camera worn by the user suffices. Nevertheless, we will see in the next section that only a combination of multiple tracking devices, the major topic of this thesis, enables robust and accurate tracking.

1.1.3 Sensing Technology

The sensing technology used in Augmented Reality systems is heavily influenced by both the real time interactivity and the correct registration requirement.

Key Requirements. Depending on the specific application, either *low latency or lag*, i.e. the time delay between an object's state in the real world and when information about this state is available to the computing system, or *high accuracy* is at the center of attention. The connection between latency and accuracy has been evaluated by Holloway [51]. At standard working distance in HMD-based AR setups, a latency of 1 ms will result in an error of 1 mm in the worst case.

Number and Quality of Sensors. As sensing technology fulfilling these requirements is very complex and usually expensive, common AR setups track a very limited number of objects, usually less than ten, simultaneously. The accuracy of state of the art trackers can be as high as millimeter positional and fractions of a degree orientational standard deviation.

Measurement State Space. To categorize sensors, we need the concept of *measurement state space*. It signifies the state of some real world entity that a specific sensor can estimate.

The number of *degrees of freedom (DOF)* is the most relevant descriptive criteria of the measurement state space. For example, position in three dimensional space has three degrees of freedom.

Spatial state can not only be measured in *absolute* coordinates with regard to a given coordinate system, *relative* coordinates with regard to the object's absolute state at a given point in time are common as well.

Finally, not only the direct spatial state such as position and orientation can be estimated by sensors, but as well first or higher order derivatives, e.g. speed or acceleration.

In typical AR systems, the following measurement state spaces occur. Section 2.5 gives specific information on available commercial and research trackers, this list is just for specifying the requirements AR applications have with regard to spatial state sensors.

6 DOF Absolute Pose. Sensors with this measurement state space are the standard equipment of AR applications. They estimate both the *position* and the *orientation* of an object in three dimensions, resulting in six degrees of freedom. The combination of position and orientation is referred to as *pose* in the remainder of this thesis. Examples include most vision systems such as ARToolkit and ART dTrack and typical VR equipment such as magnetic trackers from Ascension Technology.

3 DOF Absolute Position. This class of sensors is of minor importance in indoor applications, as most trackers used for AR estimate the orientation as well. However, with the Global Positioning System (GPS) and its extensions belonging to this class, it is the major workhorse for current mobile outdoor AR applications.

3 DOF Relative Position. Another class of minor importance. Relative position can be estimated by twofold integration of an object's acceleration. Accelerometers are very cheap sensors and can therefore be deployed in vast amounts compared to the more expensive sensors just described. However, due to integration, small errors in accelerometer readings sum to large errors in position. Consequently, this class of sensors can only be used in combination with other sensing modalities.

3 DOF Relative Orientation. Another important and often used class of sensors. Being based on the gyroscope principle, these have the advantages of being relatively cheap, small, have a high update rate and work in a self-contained fashion, i.e. no additional infrastructure such as fiducial marks are needed. As gyroscopes measure the angular velocity, their output has only to be integrated once to give an estimate of relative orientation. In consequence, the error characteristics are much better than with accelerometers, but still similar error accumulation drawbacks occur. Most often, 6 DOF absolute pose sensors with a rather low update rate are enhanced in their orientational output by high update rate 3 DOF relative orientation sensors.

3 DOF Absolute Orientation. With several commercial products available, this sensor class typically combines 3 DOF relative orientation sensors with magnetometers to define a reference coordinate frame based on the earth's magnetic field. Magnetometers are susceptible to field distortions caused by ferromagnetic substances or electronic equipment, however, these distortions can be corrected by the relative sensing facilities to some extent. Commercially available examples include the InertiaCube from Intersense and the MT9 from XSens, that both integrate three gyroscopes, three accelerometers and three magnetometers in a small form factor.

Note that virtually all AR applications compute 6 DOF absolute pose out of the available sensors in order to correctly augment the user's environment. In summary, the following characteristics describe the requirements for current AR sensing systems:

- support for large number of yet unknown applications;
- high accuracy;

- low latency;
- small number of tracked objects; and
- 6 DOF absolute pose state space.

1.2 Ubiquitous Computing

Inspired by computers getting ever smaller, Mark Weiser envisioned and coined the term of *Ubiquitous Computing (ubicomp)* [130]. As with AR, ubicomp is somehow related, but at sharp contrast with Virtual Reality. For VR, the major goal is to put the user in a completely artificial world. For ubicomp, the goal is to enhance the user's experience of the real world by bringing "invisible" computers into it.

1.2.1 Definition

Ubiquitous Computing can be defined best by looking at it as a logical successor in the sequence of computing paradigms. In the beginning, there were very few mainframe computers and many users per computer. This era was ended by the personal computers, where every user had his own machine. With ubicomp, a magnitude of computers is available for every user.

All these computers should become virtually invisible in their users' lives. This would be a major paradigm shift, as today's computers force the user to adapt to a virtual world consisting of windows, icons etc., instead of supporting the user almost unconsciously. Weiser takes writing as an example of such an invisible information technology. He states [130]

> The constant background presence of these products of "literacy technology" does not require active attention, but the information to be transmitted is ready for use at a glance.

If a computing system should support the user unconsciously, it has to gather knowledge about the situation the user currently is in, which is referred to as the concept of *context awareness* [105]. Context awareness is crucial to make ubicomp systems work [29]. Within this thesis, I follow the definitions of Dey [34]. He defines context and context awareness as follows:

> 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.

> A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

Ubicomp applications can only blend into the background if they are as much context aware as possible. Only then they can take away the user's burden to explicitly *tell* the system what to do. Instead, the applications try to find out what the user currently *wants*, leading to *implicit* instead of the common explicit human computer interaction [107]. Note that location of users and real and artificial objects is one of the most important parts of contextual information.

It should be noted that the term *Pervasive Computing* is often used as a synonym for ubicomp. This thesis follows Mattern's distinction [79], stating that the term Pervasive Computing was coined by industry and the technology behind aims at short term applications and business models of omnipresent computers, whereas the term ubicomp is used in a more academic and idealistic meaning, signifying a mere technical vision of user-centered computers.

1.2.2 State of the Art

Research in ubicomp is much broader than in the rather specific area of AR. As such, only some subareas of ubicomp that are of specific importance to the work described in this thesis will be highlighted.

- As with AR, researchers are exploring a multitude of possible application areas for ubicomp systems. It is not yet clear which applications or even application areas will be successful in the future.

- Many people work on inventing, implementing and evaluating user interface metaphors for ubicomp environments. Specific problems lie in how to deal with contextual information and the inherent three dimensionality of ubicomp applications.

- Ubicomp applications are often assembled spontaneously, as it is not known a priori which computing devices will be available at the moment the user issues a command. As such, much work is put into defining ad-hoc connection protocols and wireless communication techniques.

- Mobile ubicomp devices, which constitute the vast majority of ubicomp devices, are small and should blend into the background. Naturally, power management becomes an issue, as a ubicomp system needing power cables or constant replacement of batteries is not as "calm" as it should be.

- Sensor data in ubicomp applications is seldom used as is, most often it gets interpreted and semantically enriched. Finding ways to organize and estimate such high-level context data is a major focus.

It should be noted that the concept of *wearable computing* has gathered much attention over the last years. Its goal is to weave computers into the ordinary clothing of the user. Within the scope of this thesis, wearable computing and its sensing technology can be regarded as an ultra-mobile variant of ubicomp.

1.2.3 Sensing Technology

According to the survey of Beigl et al. [16], the sensors in ubicomp systems differ widely and are used for estimating many categories of context beyond location [108]. Nevertheless, one third of the sensors used in present ubicomp applications are for detecting movement or proximity, both being strongly correlated to location.

Key Requirements. As discussed above, the most important properties of typical mobile ubicomp sensors are a *low power consumption*, a *small size* and a *low price*. Otherwise, it would not be possible to let them disappear in the user's environment. The situation differs for stationary sensors like active floors or the Bat system [2] that may have a high deployment cost and large extension. In this respect, they are similar to typical AR sensors.

Number and Quality of Sensors. In contrast to AR setups, typical ubicomp applications employ a very large number of sensors. Being relatively cheap, this becomes feasible. Usually, accuracy is not a high priority issue, as the sensor data as such is usually not used directly, most times it gets refined and aggregated before being used as high-level contextual data. In addition, the ubicomp key principle of highly distributed computing favors this approach—many distributed mobile and stationary sensors delivering rather inaccurate information can be aggregated to valuable and reliable contextual information sources.

Measurement State Space. Again, the measurement state space of ubicomp sensors differs widely from the sensors used in AR applications. In general, the energy consumption, size and cost criteria lead to mobile sensors that are self-contained, i.e. do not need special external infrastructure to work properly [109]. Besides that several classes of typical ubicomp sensors can be distinguished.

Accelerometers. These are the workhorses of many ubicomp applications—they are cheap, deliver high update rates (up to 1kHz) and analogue output, and can be integrated in virtually every device due to their small size. Of course, their spatial state space is rather limited, usually to a single degree of freedom

second derivative of position, although it is easy to combine two or three accelerometers to get two or three degrees of freedom.

Movement sensors. With ball or mercury switches being the most popular sensors of that class, the state space of accelerometers gets restricted to a binary decision—whether the sensor's acceleration is beyond a certain threshold or below. For many ubicomp applications, these sensors are used to switch intelligent objects to an active state as soon as they are moved.

Proximity-based sensors. With radio frequency ID (RFID) tags being the most prominent representative, this class of location sensors is often used to get semantical information such as "Karl-Rüdiger has entered the room." Their state space can be characterized as binary 3 DOF absolute position, at the moment a RFID reader detects a tag, the tag's 3D position (then being the same as the reader's) is known with relatively high accuracy. If the RFID reader does not detect a tag, we can only make assumptions about its position, based on a movement model and the time since the last detection.

3 DOF absolute position. This state space is used by outdoor ubicomp applications that employ GPS as sensing technology. In addition, several stationary indoor tracking technologies such as the smart floor [90] or the Bat system [2] output 3 DOF absolute position.

In summary, the state space of location sensors in ubicomp systems is not homogeneous at all. Depending on the specific application, many diverse sensors are employed in diverse fashions. Still, some characteristics of ubicomp sensors should be supported by a system aggregating them:

- support for a large number of diverse applications;
- potentially very high number of sensors and sensed objects;
- high flexibility and dynamics in sensor network configuration;
- largely varying measurement state space; and
- semantically enriched location information.

1.3 Combing Augmented Reality and Ubiquitous Computing: Motivating Scenarios

The main vision behind the work described in this thesis is that many applications from the realm of ubicomp could be enhanced with user interfaces inspired

by techniques used in AR. To motivate that claim, this section gives several visionary scenarios illustrating what such systems could look like. The key ideas of the scenarios have been developed jointly with Joseph Newman from Technische Universität Graz.

For every scenario, it is assumed that tracking information is available to the envisioned applications at every point in time in space. Whenever an application needs information about the spatial state of some user or real-world object, it gets it in a sufficient quality.

1.3.1 Scenario: Emergency Situation in Smart Building

Imagine a smart building being equipped with a multitude of ubicomp sensors of varying accuracy. The building permanently monitors the location of the people living and working in it. In normal operation, the building uses this information for tracking down people if visitors are looking for them or to notify them when they come by a colleagues' office that he or she wants to talk to them [2]. In the case of fire, the tracking data of the building will be combined automatically with equipment brought to the building by emergency personnel. This aggregated information can then be used to save lives in multiple ways:

- Based on the current distribution of people, the building calculates optimal escape routes such that all people can leave the building as quickly as possible. Ambient displays and/or equipment worn by individuals such as HMDs gets used to communicate the current ideal fire exits. In addition, characteristics of users such as disabilities can be taken into account—a person in a wheelchair is guided via another route than a non disabled person.

- The sensor data can be used by firefighters to track down injured or captured people, both on a "world in miniature" view of the building for the commanders and in personal navigation aids for front line firefighters. Obviously, this increases also the firefighters' own safety, as colleagues in trouble can be detected easily.

- With AR user interfaces, firefighters can be given navigation hints in a very unobtrusive fashion. As noted by Jiang et al. for an existing research prototype [57], for users of such mission critical applications, "interacting with a device is often not their primary focus". With ideal AR systems, navigation aids and other information is just attached to some real objects such as walls or doors, making it often unnecessary to interact. In addition, AR can be used to help navigation in areas of low visibility due to power failures or smoke.

- Messages from firefighters such as "the room I'm in just collapsed" can be annotated with location information automatically. This significantly reduces the communication load.

1.3.2 Scenario: Museum Tour

A museum is a special form of a building: it is very open to the public, a large number of visitors get attracted by it and are happy to take a tour through it. It is also possible to hand out special equipment to visitors on entry. In addition to the emergency facilities described in the last section, a museum can make use of a solid tracking infrastructure in multiple ways:

- Visitors can be guided through the museum dynamically, depending on many parameters. A visitor's interest profile can be evaluated, it can be taken into account what he has seen already or how many other visitors are currently lining up in front of new attractions.

- Especially in large museums, it is often difficult to stay together in larger groups or families. Using the available location information makes rejoining friends or family members easy.

- If varying levels of tracking accuracy are available, the dynamic integration of new museum presentation techniques such as the Virtual Showcase [20] gets facilitated. For such applications, the low accuracy of typical ubicomp sensors is not sufficient, high accuracy AR sensors must be integrated dynamically.

1.3.3 Scenario: Factory Floor

The vision of having good enough tracking information everywhere is of particular interest in industrial applications–applying sensor technology in well-structured environments such as factory floors is easier than in rather unstructured environments such as museums or private buildings. Still, the prohibitive cost of high accuracy sensors for AR makes it impossible to equip a whole factory floor with suitable tracking. With a highly dynamic combination of inaccurate stationary sensors and accurate mobile sensors worn by users, several new applications are becoming feasible. In addition, a unified access to all tracking data decreases the development time and cost of new applications significantly. The following ideas illustrate the potential of "tracking everywhere" on factory floors:

- Similar to the smart building scenario detailed above, workers could be guided through the building by enhanced crowd control mechanisms; in case of emergency, specific information such as the current location of toxic substances would be provided to firefighters.

- Several already existing AR based applications (e.g. [37]) could be operated by a worker without the need to change his equipment. This reduces the amount of time spent for some time-consuming calibration tasks such as HMD calibration.

- Planning of new factory layouts could be enhanced, by moving a prototype product through the not yet existent production line. AR would be used to visualize the production line, showing potentially narrow zones and other obstacles. This is only possible by a flexible combination of the user's high accuracy mobile sensors with the coarse stationary sensors.

1.4 Problem Statement: Ubiquitous Tracking

The vision behind the work described in this thesis is a unified access to location sensor data of a broad range of sensors typically used in ubicomp and AR systems. In this section, the problem attacked and the key requirements for solutions to be developed are formally stated.

1.4.1 Problem Statement.

The problem of Ubiquitous Tracking (Ubitrack) *consists of providing an abstraction from sensors estimating the spatial state of arbitrary objects in the real world. The abstraction should be accessible by multiple applications simultaneously and handle all details of the physical and logical distribution of sensors, dynamic changes and sensor fusion transparently to the applications. Among the supported applications should be typical Augmented Reality and Ubiquitous Computing systems. The abstraction should support environments with many heterogeneous sensors that change dynamically as mobile users wearing sensors roam around.*

1.4.2 Requirements.

The last sections have discussed in some detail which sensing requirements typical applications from both AR and ubicomp have. This section describes a summary of them on a very abstract level, following Brügge's categorization of requirements [26].

Functional Requirements describe the interaction between the Ubitrack system and its environment independently of its implementation:

Provision of 6 DOF spatial data. The Ubitrack sensor abstraction has to provide data about the spatial relationships between objects in six degrees of freedom, otherwise typical AR applications would not be possible. In addition,

the system must support other measurement state spaces as well, especially the state spaces relevant for ubicomp applications as discussed in section 1.2.3.

Aggregate multiple sensors. It must be possible to aggregate the spatial data delivered by multiple sensors transparently to the application.

Provision of accuracy information. The Ubitrack system must deliver information not only about the spatial relationships between objects, but as well data describing the quality of the measurement processes that led to the estimates of spatial relationships. This accuracy information must also be available for aggregated sensor data.

Handle multiple applications. Several distinct applications must be handled simultaneously. It must be possible for the applications to influence the strategy of sensor fusion.

Ad-hoc connection of new sensors. New sensors brought into the system by e.g. mobile users must be connected in an ad-hoc manner. It must be possible that applications already running use these new sensors.

Ad-hoc connection of new sensor networks. It must also be possible to merge multiple already connected sensor networks (e.g. multiple sensors worn by a mobile user coming into a stationary environment with its own sensor network).

Dynamic Reconfiguration. The system must adjust to changing environmental conditions such as a varying set of sensors or applications dynamically. It must reconfigure itself automatically without the need of manual intervention of a system administrator or user.

Non-Functional Requirements describe the user-visible aspects of a system that are not directly related to its functional behavior:

No central component. The system should not rely upon a central component. This facilitates the ad-hoc connectivity of new sensors or sensor networks and enhances the robustness of a sensor network in case of emergency.

Real-time characteristics. The system must not introduce a latency that destroys the real-time characteristics of AR applications. Beyond the computation time for sensor fusion, no runtime overhead compared to prearranged fixed tracking systems should occur.

Scalability to large number of sensors. The system should scale well to be able to incorporate the potentially thousands of ubicomp sensors deployed in intelligent environments.

Best-effort approach. The system does not have to provide the best possible solution to a specific spatial relation need of an application. To enhance the computational efficiency, a best effort approach suffices.

1.5 Thesis Contribution

The contribution of this thesis is an abstraction from networks of sensors for location estimation. Both a formal model and a set of implementation concepts suitable for both AR and ubicomp applications are presented.

The formal model has been jointly developed with Joseph Newman from Technische Universität Graz and focusses on scalability and generality to a degree that allows the unified modeling and description of all location sensor networks described in current AR and ubicomp literature. The implementation concepts put a strong emphasis on highly distributed solutions to several problems arising when trying to apply the formal model to real world sensor networks.

In consequence, the thesis statement is as follows:

> Networks of multiple sensors for location tracking in both Augmented Reality and Ubiquitous Computing applications can be abstracted such that multiple applications can access diverse aggregations of sensor data simultaneously in an efficient manner. This abstraction can be done using the following components:
>
> 1. A graph-based formal model to describe all kinds of sensor networks in a consistent fashion.
> 2. A distributed implementation concept that allows to algorithmically convert a sensor network description based on the formal model to a network of software components running on a computer network.
> 3. A software architecture that allows the ad-hoc connection of mobile sensor networks with stationary or other mobile networks.
>
> An integrated middleware containing these components facilitates the reusability, modifiability and extensibility of multi-sensor AR and ubicomp applications.

Besides a detailed discussion of the formal model and the implementation concepts, the viability of the formalism by applying it to several representative multi-sensor setups is shown. The core parts of the distributed implementation concept have been shown to fulfill the real time requirements by simulation, and the software architecture for merging sensor networks has been shown to work in practice by means of a proof of concept application.

The work of this thesis has been developed in the context of the DWARF Augmented Reality framework [12, 13] and extends it in the problem domain of tracking. Parts of this work have been published previously in [14, 78, 88, 87, 124, 126, 127, 128, 125].

1.6 Thesis Outline

The thesis starts with an overview of the related work in several areas (chapter 2). First, applications from AR, ubicomp, distributed and context aware computing are discussed, giving an idea of the scope of my work. Second, existing sensor and sensor fusion technologies are presented and finally existing systems that provide an abstraction from individual sensor data are evaluated.

Chapter 3 is about the formal model for Ubiquitous Tracking. The concepts of *spatial relationship graphs*, *measurement attributes* and *evaluation functions* are motivated and discussed. Several examples of increasing complexity show how the formalism can be applied to real-world multi sensor tracking problems.

Chapter 4 goes on with a distributed implementation concept that allows to map the Ubitrack formalism onto a network of computers organized following the peer to peer paradigm. The distribution strategy of knowledge about the sensor network among the participating computers, how sensor data can be aggregated in a generic fashion and results of a simulation of the core algorithms are given.

Chapter 5 describes how the distributed implementation concept plays together with the concepts of the DWARF framework. The extensions developed for that purpose are detailed and results on the runtime behavior of a proof of concept implementation are given.

Chapter 6 treats the problem of connecting new sensors or sensor networks to existing networks. A DWARF-based software architecture that allows to specify configuration protocols for using generic tracking hardware in new environments is described. The architecture employs contextual information to reduce the computational complexity. Finally, several representative scenarios show how the architecture gets used for connecting multiple Ubitrack setups dynamically.

Chapter 7 presents *Ubiquitous SHEEP*, a demo application showing all parts discussed in the previous chapters working together.

Finally, chapter 8 gives a summary, draws conclusions and gives an extended list of the next steps that have to be taken to bring the Ubitrack concepts presented in this thesis into real applications.

CHAPTER 2

Related Work

Overview

The purpose of this chapter is twofold: first, it tries to give an overview of the areas of research influencing this thesis. These include Augmented Reality in general, ubicomp, context aware and distributed computing. It is discussed why this work can be of use for all kinds of AR and most ubicomp applications. In addition, some ideas how it can help context aware computing are presented.

The second part of this chapter gives an in-depth overview of sensor technology used in present AR systems for pose estimation. Advantages and drawbacks of the main tracking technologies used today are analyzed: mechanic tracking, acoustic tracking, magnetic tracking, vision-based methods using artificial and natural features, inertial sensors, gyroscopes, cell-based trackers, magnetometers and the Global Positioning System (GPS).

Related work on combinations of single trackers is described. This includes a presentation of the mathematical concept of a Kalman filter and its application to the problem of tracking for AR, and a discussion of existing error models used to describe the accuracy of an individual tracker or a combination of multiple trackers. In addition, approaches of static combinations of multiple trackers described in literature are presented and the commonalities and differences between these systems are described.

2.1 Augmented Reality Applications

The dominant application areas of AR discussed in literature have been extended over time. This section tries to give a brief overview of relevant work that shows where the work described in this thesis might contribute.

In a 1997 survey paper, Azuma [8] discusses medical, manufacturing and repair, annotation and visualization, robot path planning, entertainment and military aircraft applications for AR systems. For all these areas, Azuma states that the sensing technology has to be extended in input variety (to conceptually everything not detectable by human senses), input bandwidth (i.e. information not only about a few tracked objects, but massive information about the whole environment the user is in), accuracy and working range. He also proposes that this goal could be achieved by hybrid approaches fusing multiple sensors. In a later survey, Azuma et al. [7] present several approaches in this direction, most of them will be discussed in section 2.8. In the same paper they give three major new application directions that hold true until today: mobile, collaborative and commercial AR. Commercial AR is treated as a separate category, as the survey understands augmentations in sports events broadcast over TV networks as commercial AR. Mobile AR applications need long-range multiple target tracking data, whilst collaborative applications profit from the possibility to integrate new mobile users ad hoc. Both functionalities can best be provided by a middleware as presented in this thesis.

The first mobile AR system was proposed by Rekimoto [101, 102], and consisted of a portable tracked TV screen attached to a workstation computer via cables. In 1997, Feiner et al. presented the Touring machine [39], the first outdoor mobile AR system. This system was later extended by Julier et al. [58] for a battlefield scenario. However, the focus was on user interface issues, for tracking the same GPS/gyroscope-based setup as with the Touring machine was used.

Azuma et al. [9] made the point that mobile outdoor AR systems need hybrid tracking to work successfully. Piekarski's Tinmith system [94] follows this position and integrates vision-based ARToolkit for hand and indoor tracking into the common GPS/gyroscope outdoor tracking setup. Still, the focus is on creating hand-tailored demo applications that do not deal with an unforeseeable infrastructure.

Mobile indoor systems can not rely on GPS as long-range tracker. Newman et al. [86] used the Bat system [2] to demonstrate AR applications in a sentient environment, Klinker [64] proposed to use mobile AR for industrial maintenance applications. However, tracking was not a focus of this project, but the distributed concepts described in this thesis would enable the envisioned maintenance applications.

Collaborative AR applications have first been proposed within the Studierstube project by Schmalstieg et al. [106, 117], with a major focus on efficient distributed visualization of shared data. Whilst initially Studierstube applications were re-

stricted to single rooms, recently Reitmayr extended the Studierstube project to mobile outdoor applications [100]. Within the Shared Space project [18, 19], Billinghurst et al. evaluated user interface related aspects of collaborative AR. All these collaborative applications suffer from the unsatisfying accuracy and range of current tracking devices and could benefit from the tracker abstraction described in this thesis.

In summary, recent AR research tries to explore the application areas of mobile and collaborative applications. Both need to handle sensor data in rather unprepared environments. This thesis presents concepts that facilitate the development of new mobile or collaborative AR applications by providing an abstraction between low-level sensor data and applications.

2.2 Ubiquitous Computing

Weiser's vision of computers blending into the background and becoming part of the user's environment [130] is radically different from common desktop-based computing infrastructure. As such, research in ubicomp has been driven by applications.

In a 2000 survey paper, Abowd and Mynatt [1] review the accomplishments in the areas of natural interfaces, context-aware applications and capture and access of live experiences. They explore ubicomp from an HCI perspective. Natural interfaces are different from the traditional keyboard/mouse interface to computers in that sensors try to capture "natural" interactions of a user with the computing infrastructure, such as speech, pen-based writing, video and similar data. A major problem with such interfaces is that new kinds of error can occur, e.g. misinterpreting handwriting or speech commands. New error recovery techniques have been developed, but are still far from perfect. Research in context-aware ubicomp applications started with the Active Badge system [129], that employed a network of sensors receiving signals from user-worn badges to get information about a user's location. This knowledge was used to route telephone calls. Since then, multiple applications employed context to make the interaction of users with a computing system more natural, but still many questions remain open. The next section discusses context-aware computing in greater detail. Capture and access applications try to support users in remembering details of their lives, such as personal conversations or visual impressions, by storing data from sensors such as a video camera and indexing and organizing this information in an intelligent fashion. The focus of research has been mostly on technology that allows the handling of such large data sets, and only very few researchers have put thoughts on an automated structuring. For example, Clarkson [31] employed similarity measures on preprocessed wearable sensor data to link events within video and audio streams automatically. Abowd and Mynatt state several ubicomp research directions: Designing a continuously present computer interface, presenting informations at different levels of

the periphery of human attention, connecting events in the physical and virtual worlds and modifying traditional HCI methods to support designing for informal, peripheral and opportunistic behavior. These research directions hold still true, as can be seen by scanning through last years' Ubicomp conference series[1] and the list of ubicomp applications compiled by Rehman [97].

Hightower and Borriello [47] give a survey of location systems used for ubiquitous computing. For this purpose, a taxonomy of location systems is defined. Sensing techniques get distinguished in triangulation, scene analysis and proximity, and the main location system properties are physical position and symbolic location, absolute versus relative position, localized computation of location to satisfy privacy concerns, accuracy, range of operation, and cost. The analyzed systems offer a wide spectrum of properties, but most (e.g. RFID tags, smart floor [90], active bat [45] and RADAR [10]) have a rather low accuracy and update rate compared to typical AR requirements. The authors of this survey paper state that sensor fusion, ad-hoc location sensing and determining the accuracy of such systems are major research challenges.

The work described in this thesis serves the technical foundations of ubicomp in the direction of these challenges. Its primary goal of providing "tracking everywhere" may support the development of new ubicomp applications that need access to reliable, redundant and highly efficient location information. Applications may use this information to derive high-level context or annotate captured data by location events.

2.3 Context Aware Computing

Context awareness [105] is a key issue in ubiquitous computing research. The "calm computing" paradigm can only work if a computing system has as much knowledge about the context in which a user interacts with an application as possible. The ultimate goal is to employ contextual information that enables the computing system to do what the user *wants* without explicitly asking for every relevant detail of a task description, leading to implicit human computer interaction [107].

2.3.1 Definition of Terms

Dey and Abowd [35] provide an overview of definitions of the terms *context* and *context-awareness* and define these terms as described in section 1.2.1. A prerequisite for any context-aware application is a structuring scheme for contextual information. As proposed by Abowd [1], this can be done along the "five W's", namely *Who*, *What*, *Where*, *When* and *Why*. In brief, a user with an *identity* is performing an *activity* at a given *location* and a specific *point in time* out of some

[1] `http://ubicomp.org/`

reason. Many problems occur if these five dimensions of context should be used in practice, ranging from pure data storage and access (recent work by Harvel et al. [46] indicates that techniques from data warehousing and data mining are suited for these problems) to defining an ontology for each dimension.

The context dimension understood best is location, it has been researched to a degree that Schmidt et al. noticed that context should be more than pure location [108]. In recent years, research has put an increasing focus on recognizing activity context information [62, 66, 73]. Still, locational information can help in determining other categories of context as well, for example, a repeated movement of a user's head from left to right and vice versa might indicate that he is currently watching a tennis match, thus deriving the activity out of location.

2.3.2 Architectures Supporting Context-Awareness

In general, every context-aware application can be structured in several layers: *Sensors* measure estimates of some properties of the user's environment. This sensor data then gets *interpreted* and *aggregated* or *fused* in order to derive contextual information. This information is finally used by the application to *trigger actions*.

Stick-e Notes. Brown et al. are among the first to describe a reusable system for developing context-aware systems, the stick-e notes platform [25]. The system employs a simple SGML-based context triggering mechanism for displaying relevant information.

It distinguishes several software components: *SeTrigger* is responsible for triggering notes. *SeShow* components perform the actual displaying, and *SeSensor* components provide context measurements to the trigger mechanism.

TEA. Schmidt et al .[109] propose within the Technology for Enabling Awareness (TEA) project a layered approach to context estimation. Raw sensor data is preprocessed to provide so-called *cues*, such as *light=70%*. All cues are put in a tuple space and aggregated by a context estimation layer to provide high-level context data such as *in a meeting*. Again, context is stored in a tuple space, from which applications have access to it.

NEXUS. The Nexus Center of Excellence at the University of Stuttgart[2] is an ongoing project aiming at the definition and realization of world models for context-aware applications with a strong focus on location. The current world model architecture is organized in three layers [89]: the bottom layer contains servers providing data sources, the middle layer contains federation components that aggregate the

[2] `http://www.nexus.uni-stuttgart.de/`

raw data and the top layer consists primarily of location-aware services. The bottom and middle layers can be accessed using the Augmented World Query Language (AWQL). AWQL is an XML language that allows to specify spatial queries with certain filtering aspects.

Nexus builds on large centralized databases that are accessed by mobile client. In contrast, the concepts described in this thesis allow a fully decentralized organization of the world model. Yet, this thesis' location sensor abstraction could be used as additional input to an existing Nexus-based system, thus combining the advantages of centralized with decentralized systems.

EasyLiving. The EasyLiving project [28] aims at an architecture supporting intelligent environments that allow the dynamic aggregation of multiple I/O devices to give a single coherent user experience.

EasyLiving puts a major focus on a geometry model of the world [27]. Applications are split in several distributed components, that store abstract service descriptions of their capabilities on a central lookup repository. An XML-based message passing middleware called InConcert is proposed to handle the communication between the distributed components.

Context Toolkit. Dey [34] describes the *Context Toolkit*[3], a software framework that gives architectural support for general context-aware applications. He proposes the following architectural building blocks:

Context Widgets are software components allowing applications to access contextual information. For example, a location sensor might be encapsulated by a context widget.

Context Interpreters interpret low-level contextual data and derive higher order context information. For example, coordinates delivered by a widget can be used to derive the room a user is currently in. Interpreters can be layered, in order to derive context in a hierarchical fashion.

Context Aggregators collect related context about an entity that an application is interested in. For example, an alarm might be triggered if person *Evil Joe* is entering the *Secret Room* doing *Nasty Things*. The aggregator's job will then be to send an event whenever a context interpreter detects that Evil Joe is in the Secret Room and another interpreter concludes that he is doing Nasty Things.

Context Services handle often used actions useful for many applications, such as turning on a light.

[3] `http://www.cc.gatech.edu/fce/contexttoolkit/`

The Context Toolkit is a promising concept for specifying and structuring context-aware applications. Its layered approach has been inspiring the distributed location tracking concepts described in this thesis. However, it varies significantly both in scope and intended use. The location sensing middleware described in this thesis focusses on handling and fusing location data, without interpreting it. The output consists of mere coordinates. As such, it could be encapsulated by a Context Widget, delivering data to Context Interpreters and Aggregators. Additionally, the Context Toolkit's current implementation focusses on handling the low-frequency event streams issued by Interpreters, Aggregators and Widgets encapsulating low data rate sensors such as an iButton[4], whilst the concepts and software described in this thesis are designed to handle the high data rates issued by typical AR location sensors.

2.3.3 Summary

The work presented in this thesis builds upon the well established approach of layering context-aware applications. It can be used as the base layer of the frameworks discussed above.

The contribution of this thesis serves as a useful basis for developing new context-aware applications, by providing a highly distributed platform for determining the locational aspect of a user's context. The determined locational information can be used not only to build location-aware applications but also to aggregate higher-level context such as recognizing the user's current activity [62].

2.4 Distributed Computing

The problem treated in this thesis is of inherently distributed nature. Sensors distributed physically and logically deliver data to a dynamic network of computers hosting multiple potentially distributed applications. In consequence, some concepts explained in this section from the area of distributed computing form the basis of my work. A good introduction to fundamental concepts and implementational challenges in distributed systems is given by Coulouris et al. [32].

Client-Server versus Peer-to-Peer. The most fundamental property of a distributed system is its organization. Schollmeier [110] gave the following definitions: In general, every participating process can provide or consume resources. In a *Client-Server network*, every participant has a clear role—it either provides resources (e.g. a HTTP daemon running on some server computer) or consumes them (e.g. a web browser running on some client computer). If all participants

[4] `http://www.ibutton.com/`

consume and provide resources simultaneously, the network architecture is called *Peer-to-Peer (P2P)*.

If removing a single arbitrary entity from a P2P network does not harm the network services, a network is called *Pure Peer-to-Peer*. In contrast, a *Hybrid Peer-to-Peer* network may contain centralized components.

The implementation of the concepts described in this thesis build upon and extend the DWARF AR framework [12, 13]. DWARF employs a hybrid P2P architecture using distributed *service managers* running on every network node. MacWilliams [77] describes the software engineering challenges arising from this approach.

Synchronization. If algorithms need to be executed in distributed systems, a key distinction between *synchronous* and *asynchronous* networks has to be made. In synchronous networks, all nodes have a global common clock, i.e. events occurring in the network can be ordered [69]. This is not the case for asynchronous networks. Algorithms only requiring asynchronous networks often have a much worse worst case running time than algorithms for synchronous networks. However, the overhead to making global clocks work can be significant such that asynchronous algorithms can be advantageous in practice.

2.5 Overview of Existing AR Tracking Technologies

This section briefly presents current AR tracking devices, the advantages and drawbacks of individual technologies, to show the potential of a dynamic combination of multiple devices. It is based on Rolland's survey [104], not discussing relatively unused tracking techniques. Bishop's tutorial on tracking [21] was used as an additional reference.

To ensure a common treatment of all technologies, we have to define three terms, *sensor*, *locatable* and *tracker*.

A *sensor* is an active piece of hardware that detects the spatial relationship between a reference coordinate system and one or several objects. Using some software, the data delivered by the sensor is made available to a computer system making use of the spatial relationship.

A *locatable* is an object that can be located by a sensor. The sensor yields the spatial relationship of the locatable relative to some reference coordinate system.

A *tracker* is a sensor used for three dimensional position and/or orientation sensing in AR or ubicomp applications.

2.5.1 Mechanical Trackers

Mechanical trackers make a rigid link between the tracker and the locatable. They use mechanical devices to estimate the spatial relationship. They can reasonably be used only in very special circumstances. Mechanical trackers are simple to build, provide high accuracy and update rates, but restrict the locatable's movement massively. They have been used by Sutherland's HMD project [116], for force-feedback devices like the PHANTOM[5] from SenseAble Technologies Inc., and for 3D digitizers essentially consisting of a robot arm with a stylus on it, such as Immersion's MicroScribe digitizer[6].

2.5.2 Time of Flight

Trackers of this class measure the time of flight of a pulsed signal between multiple places on the locatable and the sensor. Assuming that the signal speed is constant, the position and orientation of the locatable can be reconstructed from the distances obtained from the time of flight measurements.

Ultrasonic Trackers. Ultrasonic sound is used as the signal for which time of flight is measured. Locatables get small and lightweight, as only a microphone or an ultrasonic loudspeaker are needed, and if infrared signals are used to trigger the emission of the ultrasonic signal, systems can even be built wirelessly. However, distortions due to ambient noise, multipath reflections and variations in the speed of sound and a limited update rate stemming from the low speed of sound are major drawbacks of these trackers. Often used ultrasonic devices are the IS-900 and IS-600 from Intersense Inc[7].

Global Positioning System (GPS). GPS uses 24 time-synchronized satellites to emit radio signals. A receiver with a clock with an unknown bias can calculate its position once it has obtained four of these signals, with an accuracy of 10 meters. The major advantage of GPS is its global outdoor availability, but it suffers from several drawbacks. A direct line of sight to at least four satellites has to be provided, making it impossible to work indoors. Accuracy is low, although it can be increased to few centimeters using real time kinematic (RTK) differential GPS, employing additional ground stations and measuring carrier phase on two frequencies. GPS is widely used, most often for military or traffic navigation applications.

[5] http://www.sensable.com/products/phantom_ghost/phantom.asp
[6] http://www.immersion.com/digitizer/
[7] http://www.intersense.com/products/

2.5.3 Inertial Sensing

Inertial sensors make use of the inertia of masses. This can be done either to conserve the axis of rotation for measuring rotation, or to measure the linear acceleration of an object.

Mechanical Gyroscopes. Gyroscopes use a spinning wheel and makes use of the conservation for angular momentum. The spatial relationship of the spinning wheel relative to the reference frame can then be measured directly. An alternative approach uses the effect of precession and measures the torque on the axis of rotation. Both approaches can only measure the rotation in a single dimension. To get a full three-dimensional measurement state space, three gyroscopes have to be combined. Gyroscopes can be built in a very small form factor, they provide a high update rate and do not need external infrastructure. However, making only relative measurements they tend to drift and therefore need constant corrections by absolute sensors.

Accelerometers. Accelerometers measure the linear acceleration of a locatable in one direction, i.e. their measurement state space is a one-dimensional second derivative of absolute position. Three devices can be combined to get a three-dimensional state space. Accelerometers are relatively cheap and can therefore be deployed in large quantities. Due to the necessity of double integration for absolute position estimation, the resulting position estimates tend to drift very much. However, accelerometers usually provide a very high update rate and proved useful for many ubicomp applications.

A special case are pedometers. These devices, usually mercury or ball switches, have a binary output that gets triggered once the acceleration surpasses a certain threshold. Using pedometers for the purpose of Ubitrack is currently not under investigation.

2.5.4 Magnetic Field Sensing

Magnetic trackers use the physical effect that a magnetic field induces a current in a coil and vice versa. The strength of this current is a function of the distance to the field's generator, which is another coil. If three perpendicular coils of a receiver mounted on a locatable measure the field strength, the 6 DOF position and orientation of the locatable can be reconstructed. Two major classes of magnetic trackers exist. AC devices use alternating current in the generator coil, producing a changing magnetic field. Polhemus[8] is the major manufacturer of AC magnetic trackers used in AR and ubicomp applications. They suffer from the magnetic field

[8] `http://www.polhemus.com/`

being distorted in the vicinity of metallic objects. In contrast, DC devices use pulsed direct current to induce the magnetic field. Vicinity of metallic objects does not any more distort the magnetic field directly, however, ferromagnetic objects and other electromagnetic field emitters such as computer monitors distort the field. Ascension[9] is the major manufacturer of DC magnetic trackers.

In principle, the distortions induced by non-moving metallic or ferromagnetic objects can be removed by precomputed lookup tables, but still magnetic tracking suffers from relatively large error, especially if the tracking range is more than a meter. On the other hand, magnetic tracker locatables can be built extremely small, do not need a line-of-sight connection to the tracker and provide a very high update rate.

2.5.5 Magnetometer/Compass

Magnetometers estimate a locatable's orientation relative to the earth's magnetic field in a single direction. Using three magnetometers yields full three dimensional absolute orientation state space relative to the earth's magnetic field. Of course, a magnetometer is very susceptible to changes in the ambient magnetic field. Another source of error is that the earth field is inhomogeneous and changing over time. However, magnetometers have the advantage that they work on a global scale and are self-contained, i.e. they do not need any external infrastructure.

Magnetometers are often combined with GPS devices in complementary sensor fusion, providing full pose information. Another common combination is with gyroscopes, whose drift is compensated and accelerometers to get absolute orientation relative to the ground. Examples of such combined devices are the MT9 from Xsens[10] or the IS-300 and InertiaCube from Intersense[11].

2.5.6 Vision-Based Trackers

Up to now, almost all devices (except inertial sensors) discussed have measured quantities that humans can not detect. The main location sensor used by humans is the eye. Vision-based systems employ cameras and computer vision algorithms to estimate the spatial relationships of a single or multiple objects relative to one or more cameras.

In general, vision-based trackers offer relatively high accuracy for the price of high computational cost. However, according to Moore's Law we can expect available processing power and thus the efficiency of vision-based trackers to grow exponentially.

[9] `http://www.ascension-tech.com/`
[10] `http://www.xsens.com/`
[11] `http://www.intersense.com/`

There are three dimensions along which vision trackers can be classified: inside-out vs. outside-in, artificial vs. natural features and active vs. passive markers.

Inside-out vs. Outside-in. A vision-based tracker uses one or multiple cameras. If the cameras are mounted in a fixed spatial relationship to the reference coordinate system and detect features being attached to the locatable, the system is called *outside-in* tracker. It has the advantage that the locatable does not have to be equipped with a camera and that the positional accuracy of the estimate is high. However, the rotational accuracy degrades (see figure 3.1 for an explanation). A commercial example is the ARTrack/dTrack system from A.R.T.[12].

If the camera is mounted in a fixed spatial relationship to the locatable (e.g. on a helmet worn by the user whose head should be tracked) and detects features that are in a fixed spatial relationship to the reference coordinate frame (e.g. markers attached to a room's wall), the system is called an *inside-out* tracker. Advantages are a high orientational accuracy (see figure 3.1 for an explanation) and a higher accuracy in the camera's and thus, in the case of a user-mounted camera, the user's vicinity, which is typically the main working area. On the downside the positional accuracy degrades. Another advantage is that, in principle, the operation range is unlimited, as long as enough features can be distinguished. State et al. [113] and Koller et al. [65] were among the first to apply computer vision to the problem of AR tracking, later Cho and Neumann [30] proposed a scalable multi-ring color fiducial method. The ARToolkit[13] is perhaps the most often used AR tracking software today and specifically targeted for inside-out tracking. A more complex inside-out device is UNC's HiBall tracker [134, 135].

Artificial vs. Natural Features. Vision-based trackers work by detecting and identifying certain *features* in a camera image using computer vision algorithms. This task can be simplified if the features have a well-known shape. For this purpose, *artifical markers* such as black squares with a certain pattern inside can be used. Several artificial marker vision trackers exist, and relatively high update rates up to 100 Hz can be obtained along with high accuracy. However, a line of sight between at least a single camera and a single marker has to be maintained all the time. In the case of inside-out trackers, the environment has to be equipped with markers, which may only be acceptable in some industrial environments. In addition, this contradicts Weiser's vision of "calm", invisible computing, as the tracking infrastructure becomes clearly visible to all users.

In contrast, natural feature trackers employ structures such as corners, edges or planar shapes to estimate the camera's pose. This is a very active area of research [43, 63, 111, 112], and still in its infancy. Markerless trackers suffer primarily

[12] `http://www.ar-tracking.de/`
[13] `http://www.hitl.washington.edu/research/shared_space/download/`

from the *initialization problem*, which consists of finding an initial guess of the camera's pose when the system gets started. Compared to keeping track of the camera's subsequent movements, this is a very hard problem in general, as a potentially infinite number of initial pose guesses has to be verified or rejected, based on the natural features' properties.

The work described in this thesis helps to overcome this problem by providing a unified access to all spatial relationships available in the system. Some of these relationships can help natural feature trackers to make a sufficiently accurate initial guess about the camera's pose.

Active vs. Passive Markers. In the case of artificial feature trackers, we can distinguish two subclasses, depending on the markers: *active* markers emit (possibly modulated) light, be it visible or infrared (IR), and therefore make the recognition task even simpler. In addition, if IR markers are used, a vision-based tracker can even work in darkness. In contrast to active IR markers, *passive* markers are subject to varying lighting conditions, e.g. a system with passive markers might work indoors, but has problems outdoors in full sunlight.

2.6 Fundamentals of Sensor Fusion

The core topic of this thesis is the aggregation of spatial relationships delivered by multiple sensors. This process is called *sensor fusion* and a well researched topic in many fields of engineering and computer science.

Sensor fusion for location tracking can be divided into three classes [24]: complementary, competitive and cooperative sensor fusion.

Complementary Sensor Fusion. Sensor fusion is called *complementary* if it employs sensors which do not depend on each other. The combination of their readings gives a more complete estimate of the system state. For example, two sensors tracking the location of an object in two non overlapping areas work in complementary fashion, Also, a GPS device (3 DOF position) and a magnetometer (3 DOF orientation) attached to the same locatable complement each other and yield a 6 DOF pose state space.

Competitive Sensor Fusion. Things get more complicated if two or more sensors give an estimate of the same spatial relationship. This is called *competitive* sensor fusion. Its purpose is twofold: first, redundancy introduced by multiple measurements can be important for safety concerns. Second, sensors with varying error distributions can be fused to minimize the measurement error.

In both cases, conflicts between readings have to be resolved, which is a complex problem. The Kalman filter described in the next section is probably the most often used solution to this problem.

Cooperative Sensor Fusion. The most complex type of sensor fusion is based on a tight coupling of multiple sensors and called *cooperative sensor fusion*. A prominent example of this class is coupling multiple cameras for 3D reconstruction based on computer vision algorithms. This problem cannot be approached in general, as very much detail about the physical properties of the sensors involved must be known.

However, a common case of cooperative sensor fusion is the aggregation of spatial relationships using the fact that they are transitive: if we know the spatial relationship of object B relative to object A and the relationship of object C relative to object B, we can infer the relationship of object C relative to object A. This thesis presents a general solution for the cooperative fusion of sensors based on spatial transitivity.

2.7 Fusing Multiple Trackers: Mathematical Tools

This section discusses the mathematical background of common sensor fusion methods that are used to incorporate external knowledge about the spatial relationships measured and to fuse data from both complementary and competitive sensors. Special attention is put on the Kalman filter, which is the filter type used most often. In addition, various error models that are used to describe the accuracy of measurements are presented.

2.7.1 The Kalman Filter

The purpose of this section is to give a very brief idea of the general concepts of a Kalman filter and how it might be used for location sensing. It is not intended as reference for the Kalman filter, see the tutorial from Bishop and Welch [133] or the book from Gelb [42] for that purpose. The explanation is based on [133].

State-Space Models. In location sensing, the sensors give observable measurements that are related to the true state of the system. Thus, we get an approximation or *estimate* of the real spatial relationships we are interested in. The spatial relationship can be treated as a dynamic process over time:

$$x_k = \mathbf{A}x_{k-1} + \mathbf{B}u_k + w_{k-1} \tag{2.1}$$

The current state x_k depends on a linear combination of the previous state x_{k-1}, a linear combination of an optional *control input* u_k and *process noise* w_{k-1}.

However, the measurements might handle the internal state as a "black box", the only assumption we make is that the measurements z_k are a linear combination of the state x_k corrupted by some *measurement noise* v_k:

$$z_k = \mathbf{H}x_k + v_k \tag{2.2}$$

We assume both the process and measurement noise w_k and v_k to be random variables independent of each other. They are assumed to have zero mean and white error distribution, i.e. the underlying random process is completely uncorrelated with itself at any time except the present. They have normal probability distributions

$$p(w) \sim N(0, \mathbf{Q}) \tag{2.3}$$
$$p(v) \sim N(0, \mathbf{R}) \tag{2.4}$$

with *process noise covariance* $\mathbf{Q}$ and *measurement noise covariance* $\mathbf{R}$.

All variables ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{H}$, $\mathbf{Q}$ and $\mathbf{R}$) might change at every time step, but we assume them to be constant.

We further define two kinds of estimates: the *a priori* estimate $\hat{x}_k^-$ is based on all information up to time step $k-1$, the *a posteriori* estimate $\hat{x}_k$ is the estimate of the real state given knowledge about the measurement z_k at time step k in addition to all knowledge used for $\hat{x}_k^-$. We then define the *a priori* and *a posteriori* error estimates as

$$e_k^- \equiv x_k - \hat{x}_k^-$$
$$e_k \equiv x_k - \hat{x}_k$$

and the corresponding error covariances as

$$\mathbf{P}_k^- = E[e_k^- e_k^{-\mathrm{T}}] \tag{2.5}$$
$$\mathbf{P}_k = E[e_k e_k^{\mathrm{T}}] \tag{2.6}$$
$$\tag{2.7}$$

Discrete Kalman Filter. For the discrete Kalman filter, the measurements occur and the state is estimated at discrete points in time. The algorithm works recursively in two steps, first a *time update*, then a *measurement update*. It can be shown that the discrete Kalman filter is optimal in the sense that it minimizes the estimated error covariance $\mathbf{P}_k$.

The time update can be regarded as a prediction of the system state and the a priori error estimate and is expressed by the equations

$$\hat{x}_k^- = \mathbf{A}\hat{x}_{k-1} + \mathbf{B}u_k \tag{2.8}$$
$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q}. \tag{2.9}$$

The measurement update tries to compute a new a posteriori estimate as a linear combination of the a priori estimate $\hat{x}_k^-$ and a weighted difference between the measurement z_k and a measurement prediction $\mathbf{H}\hat{x}_k^-$. For this purpose, the so-called *Kalman gain* $\mathbf{K}_k$ needs to be computed, then the a posteriori state estimate and error covariance can be computed:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T \left(\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R}\right)^{-1} \tag{2.10}$$

$$\hat{x}_k = \hat{x}_k^- + \mathbf{K}_k \left(z_k - \mathbf{H}\hat{x}_k^-\right) \tag{2.11}$$

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}\right)\mathbf{P}_k^- \tag{2.12}$$

After each time/measurement update pair, the process is repeated with the old a posteriori estimates as inputs for the new a priori estimates. This makes the Kalman filter a *recursive* algorithm and makes practical implementations feasible. The computational cost is the same at each step. There is no necessity to accumulate all measurements obtained up to now, as with other filtering techniques.

Extended Kalman Filter. The discrete Kalman filter assumes that the process is controlled by a *linear* difference equation and a *linear* measurement-process relationship. In most real-world applications, this does not hold true; there are non-linear relations.

The discrete Kalman filter can be modified to the *Extended Kalman Filter (EKF)* to cope with such situations. Note that the EKF is just an ad-hoc approach to such problems, no optimality can be guaranteed. However, it works reasonably well in many applications. The basic idea of the EKF is to linearize the estimation around the current estimate using partial derivatives of the process and measurement functions. These functions can be expressed as

$$x_k = f\left(x_{k-1}, u_k, w_{k-1}\right) \tag{2.13}$$

$$z_k = h\left(x_k, v_k\right) \tag{2.14}$$

with w and v being the process and measurement noise. As those are often unknown, we assume them to be zero.

The process of linearization involves computing Jacobian matrices of partial derivatives of f with respect to x and w

$$\mathbf{A}_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}\left(\hat{x}_k, u_k, 0\right) \tag{2.15}$$

$$\mathbf{W}_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}\left(\hat{x}_k, u_k, 0\right) \tag{2.16}$$

$$\tag{2.17}$$

and partial derivatives of h with respect to x and v:

$$\mathbf{H}_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\hat{x}_k, 0) \tag{2.18}$$

$$\mathbf{V}_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}(\hat{x}_k, 0) \tag{2.19}$$

$$\tag{2.20}$$

The general steps for the EKF are similar to the discrete version, only the filter equations change. The time update equations are:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k, 0) \tag{2.21}$$

$$\mathbf{P}_k^- = \mathbf{A}_k \mathbf{P}_{k-1} \mathbf{A}_k^T + \mathbf{W}_k \mathbf{Q}_{k-1} \mathbf{W}_k^T \tag{2.22}$$

The measurement update equations are:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T \left(\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{V}_k \mathbf{R}_k \mathbf{V}_k^T\right)^{-1} \tag{2.23}$$

$$\hat{x}_k = \hat{x}_k^- + \mathbf{K}_k \left(z_k - h(\hat{x}_k^-, 0)\right) \tag{2.24}$$

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k\right) \mathbf{P}_k^- \tag{2.25}$$

Using a Kalman Filter in Practice. The Kalman filter has several parameters that are used for modifying its behavior:

The State Vector holds all information of the observed system we are either interested in or need for expressing the system dynamics. For example, if we want to observe a system that is moving along a single axis x, we might also be interested in its velocity v and acceleration a for describing the system dynamics, resulting in a state vector $x_k = (x, v, a)^T$.

The Process Model is governed by the linear equation system expressed in the matrix $\mathbf{A}$. For example, to model the movement just described, $\mathbf{A}$ becomes

$$A(\Delta t) = \begin{pmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{pmatrix},$$

leading to an update equation $\hat{x}_k^- = \mathbf{A}(\Delta t)\hat{x}_{k-1}$. In the case of an EKF, the process model gets expressed by an arbitrary function f.

The Measurement Model expresses how the state to be estimated results in the measurements done by the sensor and is governed by the linear system expressed by the matrix $\mathbf{H}$. For example, if the sensor gives us the object's position directly, $\mathbf{H}$ becomes $\mathbf{H} = (1, 0, 0)^T$. In the case of an EKF, the measurement model becomes an arbitrary function h.

The Process Covariance $\mathbf{Q}$ is updated at every state, but has to be seeded somehow. For a constant, we could assume $\mathbf{Q}$ to be 0.

The Measurement Covariance $\mathbf{R}$ expresses how much influence a measurement has on the next prediction/correction cycle. If we can expect the sensor to be of low quality, the covariance will be rather high, leading to a filter that is smoothing the input data. For high quality sensors with low measurement noise, the filter smooths the data less, but can react quicker to changes in the input data.

Careful fine tuning has to be done to make a Kalman filter work in practice, in general it is not possible to set it up in an automated way. As such, the sensor fusion system described in this thesis provides means to incorporate hand-tuned Kalman filters, but does not provide any general-purpose filtering.

Prediction. To make a Kalman filter work, a dynamic motion model has to be specified. This motion model is used to both update the state estimate and the error covariance in the filter's prediction step.

As proposed by Azuma [6], we can also use this model to predict the state of the tracked object at some point in time p.

Competitive Sensor Fusion. Fusing multiple sensors in a competitive fashion is conceptually simple employing a Kalman filter. The internal system state space stays identical, as does the dynamic process model. However, the measurement vector gets extended and the measurement model needs to be adapted. For example, if we have a sensor s_1 measuring an object's position along a single axis and a sensor s_2 measuring its acceleration along this axis, we have a measurement vector $z = (x(s_1), a(s_2))^T$ and a measurement model matrix

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Note, however, that this approach has the underlying assumption that both measurements have been taken at exactly the same point in time, which is rarely true in practice.

Asynchronous Complementary Sensor Fusion: Single Constraint at a Time (SCAAT). Welch [132] treated the timing problem just mentioned extensively and described the *Single-constraint-at-a-time (SCAAT)* approach to sensor fusion. The key idea is to store multiple measurement models and measurement covariances, one for each sensor to be aggregated. Whenever data from a particular sensor becomes available, the corresponding matrices are "swapped in" and a prediction/correction cycle of the filter is performed.

2.7.2 Other Filtering Techniques

The Kalman filter is the filtering framework used most often for tracking purposes. However, other techniques exist and are useful in specific areas.

The Complementary Filter. The complementary filter [11] can be seen as a very restricted Kalman filter. It is used primarily for Inertial Navigation System (INS) and GPS location sensing.

The sensor measurements $z(t)$ are modeled as the sum of the signal $s(t)$, i.e. the system state to be estimated, and some noise $n(t)$:

$$z(t) = s(t) + n(t) \tag{2.26}$$

To illustrate the operation of a complementary filter, consider the case where two independent measurements of the same signal are available:

$$\begin{aligned} z_1(t) &= s(t) + n_1(t) \\ z_2(t) &= s(t) + n_2(t) \end{aligned}$$

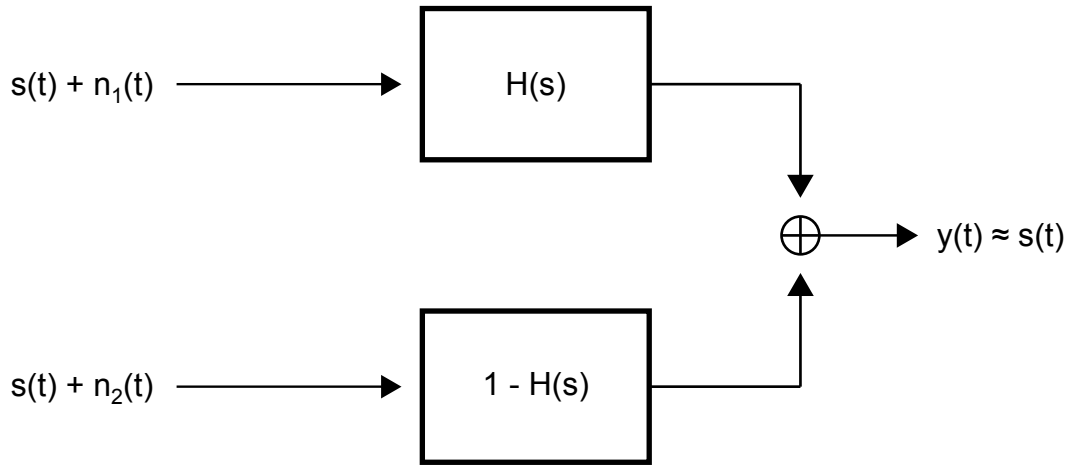


Figure 2.1: A complementary filter for two measurements (taken from [11]).

The resulting structure of the complementary filter is shown in figure 2.1. Note that the original signal passes without any modification, regardless of how $H(s)$ is defined.

The Particle Filter. The particle filter is a Bayesian technique well suited for non-linear and non-Gaussian tracking problems [5, 103]. Similar to the extended Kalman filters, particle filters provide suboptimal solutions to complex tracking problems with non-linear system dynamics and measurement noise models.

The key idea is to represent the underlying stochastic process's probability density function (pdf) as a set of samples of the state variable of interest. Multiple copies called *particles* of this variable are kept, and each is associated with a weight. The weight signifies the quality of the specific particle. The pdf estimation is the weighted sum of all particles. As the Kalman filter, the algorithm has two phases and is recursive. At each step, each particle's state is modified according to the system model (*prediction* step), then, each particle's weight is recomputed (*update* step). To simulate noise in the measurement process, each prediction step adds random noise particles. Each update step removes particle of negligible weight to limit the computation cost.

Discussion. Depending on a specific application area, a careful choice of filtering algorithms helps in reducing measurement noise. However, this choice can not be automated due to the multitude of parameters influencing every specific tracking problem. As such, the concepts described in this thesis have to provide means for integrating existing or newly developed filtering schemes for particular problems, but can not provide automatic means of setting up such schemes.

2.7.3 Error Models for Describing Tracking Accuracy

This section gives a brief overview of how tracking error can be classified, and how it can be represented mathematically.

Classification of Tracking Error. Tracking accuracy is affected by three classes of error [51].

Static error occurs even when the user's viewpoint and any objects in the environment are not moving at all. Examples of static error are field distortions of magnetic trackers or inaccuracies caused by an uncalibrated head mounted display. Static error can be eliminated by careful *calibration* [123] of systems, i.e. by comparing the measured values with a "ground truth" obtained via some alternative, potentially interactive [122], procedures. The work described in this thesis can serve as a basis for obtaining necessary alternative measurements.

Non-repeatable error (or jitter) cannot be calibrated out of the system, it is mainly caused by noise.

Dynamic error shows up as soon as anything in the system starts moving. According to [21], it can be divided in four classes. *First-oder dynamic error* is caused by the object moving between two subsequent measurements. If the *simultaneity assumption* (i.e. multiple tracker readings are assumed to be taken at the same point in time) is hurt, more dynamic error is induced. The *sensor sample rate* needs to be twice as high as the expected frequency of motion, which is in case of head and arm motion ranging from 2 to 20 Hz. If the update rate of the tracking

setup is lower, additional error gets induced. Finally, the *synchronization delay* is caused by asynchronicity between taking measurements and estimating a pose out of it. Especially in larger distributed multi-tracker setups as described in this thesis, care has to be taken to minimize the influence of this kind of delay.

Gaussian Representation of Error. To make a Kalman filter work, a Gaussian error model of the measurements has to be provided. For this reason and for sake of simplicity, the vast majority of error models discussed in current literature are based on a Gaussian distribution of some parameters describing the estimated spatial relationships. It is assumed that the reader is familiar with basic statistics of multi-dimensional random variables.

Hoff [49, 50] proposes to model error as a Gaussian distribution of a six-dimensional state vector. It contains three translational components x, y, z and three Euler angles α, β, γ. As such, a 6×6 covariance matrix $\mathbf{C}$ expresses the error. Hoff also derives equations for propagating such an error description to changed coordinate systems and/or aggregation of spatial relationships and uses the covariance matrices to visualize error by ellipsoids. This is an excellent method for representing positional error. The drawbacks of this approach stem from using Euler angles to represent orientation. The three values used to represent orientation are treated as almost independent in the error model, but depend heavily on each other [21].

Gaussian Distribution of Unit Quaternions. Unit quaternions [67] have many advantages compared to other representations of error, as they do not have singularities in their representation. For the remainder of this thesis, rotations are represented as unit quaternions. Within the efforts related to this thesis, Pustka [96] proposes to represent rotational error by "small" quaternions, i.e. rotations that can be approximated by some almost unit quaternion

$$e_r \approx (e_{r,x}, e_{r,y}, e_{r,z}, 1).$$

The complete model for representing tracking error in position and orientation consists of a normal distribution with zero mean and covariance represented by a 6×6 matrix $\mathbf{C}$:

$$(e_{t,x}, e_{t,y}, e_{t,z}, e_{r,x}, e_{r,y}, e_{r,z})^T \sim N(0, \mathbf{C}) \tag{2.27}$$

For the formulas needed for error propagation and applying this error model to an extended Kalman filter, see Pustka's thesis [96].

Other Error Representations. A Gaussian distribution is only an idealized assumption of error behavior that is useful in many mathematical tools such as the Kalman filter. "Real" error behaves differently, and some more complicated techniques for modeling it have been proposed [40].

It seems reasonable to represent location as a probability distribution. Besides the Gaussian representation just discussed, a representation as a mixture of Gaussian for multi-hypotheses tracking, grid-based approaches with a partitioning of space and particle filters have been proposed [5, 68].

Most of these approaches share a much higher computational complexity than the Kalman filter.

2.8 Fusing Multiple Trackers: Existing Systems

The idea of combining multiple sensors for location tracking is not new as such. There have been multiple efforts to provide software abstractions for a unified access to multiple sensors, and several hand-tuned indoor and outdoor systems employing sensor fusion techniques for location tracking. In addition, the robotics research community has worked in similar directions.

2.8.1 Software Abstraction Layers

In recent years, the need for a reuse of tracker drivers lead to some software frameworks that abstract the actual tracker interfaces and allow a unified access to multiple sensors.

OpenTracker. Reitmayr developed the OpenTracker [99] tracking and input device abstraction framework for the Studierstube [106] AR system. OpenTracker provides a data flow architecture that is configurable via XML. Using an object-oriented approach, the various transformation steps employed in a typical tracking setup can be modeled easily as a graph of tracking objects, conceptually similar to a scene graph used in computer graphics. The graph consists of *source nodes* that provide tracking data by e.g. a hardware driver, *filter nodes* doing some processing of the data and *sink nodes* sending the data to an application. A multi-threaded execution model allows the simultaneous use of several transformation chains. In addition, special source and sink nodes allow to use IP multicasts for distributed tracking.

OpenTracker is a very flexible library that adds almost no runtime overhead to a handcrafted solution to many tracking problems in AR. It could serve well as a runtime infrastructure for the multi-sensor approach described in this thesis. However, it currently has a static, although easily modifiable, file-based configuration and is restricted to a 6 DOF pose state space of spatial relationships.

Virtual Reality Peripheral Network (VRPN). As its name suggests, VRPN [118] has been developed primarily for Virtual Reality applications. It consists of a set of

classes within a library to be used by applications and a set of servers that interface with supported tracking hardware.

VRPN handles most commercially available VR/AR tracking devices. Recently support for integrating it with OpenTracker has been added. VRPN is not only aimed at tracking devices, but handles all kinds of input devices commonly used in VR applications, such as buttons, dials and force feedback devices.

The additional latency introduced by VRPN is almost negligible and dominated by the network latency necessary for the underlying communication between hosts.

VRPN is an excellent solution for static setups with a single or a small number of tracking devices that need to be accessed in a consistent way. No means of changing the tracking infrastructure at run time are given.

Tinmith Tinmith [94, 95] is a software architecture for outdoor Augmented Reality environments. It provides a platform that facilitates the development of new applications for virtual environments, based on principles from object oriented languages and employing a data flow model. Tinmith consists of modules running as separate UNIX processes and communicating via IP connections. As such, an application can be distributed among various hosts [92].

Tinmith provides support for all aspects of AR applications, and consequently contains a tracking abstraction layer. Various coordinate systems (latitude/longitude, UTM etc.) are supported, and a distinction between absolute and relative trackers can be made using position and orientation offset classes. To handle the potentially large coordinates arising using UTM coordinates, a transparent local coordinate system is introduced in the OpenGL-based presentation subsystem.

Tinmith is focused on high performance and provides library functions to application programmers. As such, it does not contain support for dynamic changes in the network and/or sensor infrastructure.

Location Stack. The Location Stack [48] was the first effort towards a unified abstraction of multiple location sensor setups. It was developed within the EasyLiving [28] project, which was the first project similar in scope to the combination of ubicomp and AR treated in this thesis.

The Location Stack concept proposes to model location sensing in a hierarchy similar to the ISO/OSI network stack. The Location Stack's layers are:

Sensors: Contains sensor hardware and software drivers for it.

Measurements: Contains software that transforms raw sensor readings to canonical location representation, including accuracy description.

Fusion: Contains software to dynamically aggregate measurements by ways of complementary, competitive or cooperative sensor fusion.

Arrangements: Contains software to reason about the relationships between two or more objects. These relationships are not only spatial, but can also describe properties such as containment or proximity.

Contextual: Contains software to merge location data with other contextual information such as temperature or user identity to derive high-level context.

Activities: Contains software such as machine learning algorithms to derive activities, i.e. semantic states in a ubicomp application.

Intentions: Contains software to reason about the user's cognitive desires.

The Universal Location Framework (ULF) [44] is an implementation of the Location Stack concept. It uses proximity sensors, a 802.11b wireless LAN based coarse positioning tracker and GPS as sensors. These get aggregated in a fusion component employing Bayesian filtering techniques [40, 68].

The Location Stack is the project most similar to the multi-sensor approach described in this thesis. It also uses a layered architecture to enhance reusability, and the Easy Living scenario is very similar to the intended Ubitrack application domain of intelligent environments.

However, the Location Stack currently does not address the real time and accuracy requirements of AR applications, typical sensors have an accuracy of 15 cm and a low update rate.

Architectures Supporting Context-Awareness. Recalling the discussion of context-awareness architectures such as the Context Toolkit in section 2.3.2, one might be mislead into thinking that these architectures can be used for the Ubitrack problem tackled in this thesis.

The described architectures also use sensors for estimating the location of users and objects. However, the very nature of the sensors used differs from the sensors handled throughout this thesis. For example, a typical location sensor of the Context Toolkit detects the identities of people being in a room. The update frequency of the resulting data stream is extremely low compared to a typical 6 DOF absolute pose sensor used for AR. In consequence, within the design of the discussed context-awareness architectures no special focus has been put on the scalability and efficiency issues that arise with high-frequency sensors.

Still, the work described in this thesis can be integrated easily in the discussed architectures, if the whole sensor abstraction of this thesis is treated as a single "input sensor" within the architectures that can then be used to derive contextual knowledge.

2.8.2 Indoor Systems

Indoor systems have been the major focus of AR research, as most viable applications happen indoors and the environment, e.g. lighting conditions, can be controlled much better than under open sky.

Foxlin applies Kalman filters for sensor fusion, and achieves full 6 DOF pose tracking by a combination of inertial and ultrasound sensors [41] using complementary sensor fusion. The system is commercially available as IS 600 from Intersense Inc.

Hoff [49] was among the first to combine multiple optical trackers for competitive sensor fusion. He proposed to combine an outside-in optical tracking system (Northern Digital Optotrak) with a helmet mounted inside-out camera-based tracker. Both systems track the same targets, but have varying error distributions. Error is modeled as a 6×6 covariance matrix of three positional and three orientational parameters, the latter represented as Euler angles. Hoff showed that the combination of both systems significantly reduces the overall error, and illustrated this observation with uncertainty ellipsoids.

Within the Studierstube project, Kalkusch and Reitmayr developed the SignPost application [59], a indoor navigation system. Within this application, OpenTracker is used to fuse tracking data from ARToolkit and an inertial tracker. ARToolkit is used to compensate for the drift of the inertial tracker and the inertial tracker stabilizes ARToolkit's rotational readings. As such, SignPost employs cooperative sensor fusion.

As mentioned above, natural feature trackers are currently a very active area of research in the AR community. Current algorithms are quite good at tracking objects from frame to frame. However, if the mobile camera of an inside-out approach gets rotated too fast, motion blur occurs and the tracker needs to be reinitialized. Klein and Drummond [63] overcome this problem by fusing data from high-rate gyroscopes and a natural feature based optical tracker. They use motion prediction based on data from the gyroscopes to enhance the quality of initial guesses the optical tracker makes. Still, the initialization of the natural feature tracker, i.e. getting an initial guess of the camera's pose remains a problem. Najafi et al. [84] propose to use coarse estimates of the camera's 3D position in combination with environment maps for automated initialization. The work described in this thesis could serve as a suitable software infrastructure providing such coarse estimates whenever they are necessary.

Within ubicomp research, sensor fusion for location tracking has not been investigated systematically, only the EasyLiving project [28] discusses some issues occurring when multiple sensors are used simultaneously. EasyLiving employs the Location Stack described above to model the sensor network, and uses stereo vision and color blob tracking as sensing modalities. However, no real sensor fusion is done, different systems track different objects, and aggregation of their results is

done on a semantical level [27].

None of the systems described has been designed for a dynamically changing sensor infrastructure, which is the major design goal of the work described in this thesis.

2.8.3 Outdoor Systems

The Touring Machine developed by Feiner et al. [39] was the first system to combine aspects of Augmented Reality and mobile computing. The application consists of a mobile computer worn by a user roaming around a campus. A head-mounted display is used to overlay textual labels on campus buildings, and both a handheld computer and a gaze-oriented interface are used to control a navigation application. Tracking is done by fusing data from a magnetometer/inclinometer combination for orientation and a differential GPS (DGPS) receiver for position in a complementary fashion. This simple but robust and wide range tracking setup has become the de facto standard for most outdoor AR systems.

You, Neumann and Azuma combined gyroscope and compass tracking with vision algorithms in a cooperative fashion [137, 138, 136] to enable a more robust outdoor tracking application.

Piekarski's Tinmith system [92] uses a DGPS receiver in combination with a 3-axis digital compass to measure the user's position. This system served as the basis for the ARQuake outdoor AR gaming system [93, 119]. In later Tinmith versions [95], an ARToolkit based tracker was used to track the user's hand relative to his head, in order to provide an intuitive user interface.

Reitmayr and Schmalstieg use a very similar approach to explore collaborative AR in outdoor navigation and information browsing applications [100]. Again, the focus is not on sensor fusion, but rather on user interface and system architecture issues of mobile outdoor AR applications.

Our own DWARF pathfinder application [12] employed a combination of a 1-DOF compass and a GPS device to track a user's position outdoors, and also did not focus on real sensor fusion. Instead, the possibilities of a distributed component-oriented approach to AR were explored.

In summary, sensor fusion has not been a major focus of research in outdoor and mobile AR or ubicomp systems, as most applications' requirements can be satisfied by a combination of some orientation sensor with a GPS receiver. In consequence, the work described in this thesis does not focus explicitly on outdoor applications. However, the mobility of its users is at the center of attention. Particularly the ad-hoc combination of mobile users' tracking devices has been a major area of work.

2.8.4 Robotics Systems

In robotics research, location tracking is an active area of research. Yet, the connections to research in AR and ubicomp are very limited for various reasons. Tracking robots is much simpler than tracking people, as a robot can be controlled fully from outside. The range of operation of a robot is often foreseeable, however, there have been some research projects that aimed at calibrating robots in an autonomous fashion in unknown environments.

Kelly [61] proposes to use appearance mosaics in flat visually textured surfaces for mobile robot guidance. Meng and Zhuang [80] present a method to calibrate a robot-arm mounted camera's intrinsic and extrinsic parameters up to a single scale factor out of multiple calibration targets at unknown locations. Both methods, as well as similar research efforts in robotics, are not applicable to the problems discussed in this thesis without major modifications, as their processing speed is much too slow for real time operation.

A key issue in robotics research is how to build maps of unknown environments and use them for navigation concurrently. This problem is commonly reffered to as *Simultaneous Location and Mapping (SLAM)* and can be naturally applied to decentralized distributed robots. For example, Feder et al. [38] use sonar for SLAM, with the goal of navigating autonomous underwater vehicles.

In recent years, multisensor systems for robot localization have become a topic of major research. Mutambara and Durrant-Whyte present a transputer-based fully decentralized solution to data fusion and control problems in modular wheel mobile robots [82]. Brooks et al. [23, 22] built a system of distributed sensors for robot and people tracking. As with Mutambara, the focus of their work is on tracking algorithms and not on a decentralized architecture and abstraction layer as treated in this thesis. Yet, the tracking ideas described in robotics will be valuable additional input modalities for this thesis' concepts.

CHAPTER 3

A Formal Model for Ubiquitous Tracking

Overview

This chapter presents a theoretical formalism that allows us to model arbitrary sensor networks. It forms the base of the sensor fusion middleware discussed in the remainder of this thesis. The formalism has been specified in collaboration with Joseph Newman from Technische Universität Graz, Austria. We call this model and the efforts built on top of it *Ubiquitous Tracking (Ubitrack)* [87, 88, 128].

The goal of the Ubitrack model is to provide, at any point in time, an optimal estimate of the spatial relationships between arbitrary objects. How optimality is defined, depends on the specific application interested in a particular relationship.

The formalism employs a graph-based model of spatial relations in the real world to express available information from sensors of all kinds in a consistent way. To introduce the formalism, properties of real-world relationships, which we are finally interested in, are discussed.

Unfortunately, measurements derived from sensor data are just an estimate of the real world's state. The properties of this estimate are described using a well-defined set of *attributes*, and several possible choices are presented. The most simple way of combining multiple sensors is to make use of the transitivity of spatial relations. Such combinations can be treated in the same way as raw measurements, but care has to be taken that the attributes describing such combinations must be computed using well-defined rules, which will be discussed for the presented attribute sets.

Application dependent *evaluation functions* work on the attributes of measurements. They are used in a general concept of how knowledge can be inferred from measurements. The result is a model of the real world that gives every application

the best possible estimate of available spatial relationships.

To illustrate how the formalism is applied to real applications, several examples of increasing complexity are given, that indeed show how seemingly highly distinctive applications can be handled with the same tracking abstraction.

3.1 Design Goals

Reconsidering the requirements listed in section 1.4.2 for an overall Ubitrack solution, we start with a discussion of the goals that led the design of the formalism presented in this chapter.

Implementability. The Ubitrack formalism is not standing in and of its own. Besides offering the possibility to describe multi-sensor setups in a uniform fashion, it should also serve as the basis of a real-world implementation. Therefore, every design decision had to be checked for practical implementability.

Extensibility of Application Domains. The main requirement for the formalism is that it allows applications to get optimal estimates of two objects' spatial relationship. For typical AR applications, optimality is defined by a combination of low latency and high pose accuracy. However, new applications might be developed that favor low monetary cost over high pose accuracy or define optimality in a yet unknown sense. It must be possible to extend the formalism at a later point in time to incorporate such unforeseen applications.

Extensibility of Measurement State Space. Although the work described in this thesis mainly deals with 6 DOF sensors suitable for AR, the formalism must be capable of handling sensors with different measurement state spaces, varying both in the degrees of freedom and in the number of derivatives of spatial relationships. This ensures that all effort put into the formalism can be reused when extending an actual implementation to large-scale ubicomp environments with a vast number of diverse sensors.

Reusability of Existing Systems. As discussed in section 2.8, many systems doing sensor fusion already exist. They usually are hand crafted for specific applications. It is advisable to reuse the massive amount of specific knowledge that went into their implementations. As such, the formalism must be capable of integrating such solutions to particular tracking problems without breaking its generality. In addition, it must be possible to integrate new tracking devices without major efforts.

Focus on Time. Location tracking is a time critical issue. Filter systems such as those described in section 2.7.1 allow to estimate two objects' spatial relationship in some point in the future. Large databases may also store past relationships. The formalism should be able to handle all these time-related issues in a uniform fashion, without hurting the real time capabilities necessary for AR applications.

3.2 Spatial Relationship Graphs

The Ubitrack formalism consists of several components that build upon each other. In this section, all will be discussed in detail, based on several examples.

3.2.1 Fundamentals

In their work for the EasyLiving Geometric Model (EZLGM), Brumitt et al. [27, 28] propose to use *entities* as base items, representing the existence of an object in the real world. In addition, *measurements* are used to describe the pose of an entity relative to another. Entities can have a polygon-shaped *extent* describing their physical expanse, and measurements have an *uncertainty* associated with them. The measurements in the EZLGM then describe an undirected graph with vertices corresponding to entities' coordinate frames and edges corresponding to measurements.

The Ubitrack formal model builds upon these ideas. A directed *Spatial Relationship Graph (SR graph)* is the basic building block of the model. Real or virtual objects are represented as *nodes* in this graph, whereas spatial relationships between objects are represented as *edges*.

The formalism does not enforce a particular spatial state space. Throughout this thesis, we will primarily use the 6 DOF pose state space, which is typical for AR systems. Other possibilities can be modeled as additional edges in the SR graph in the same manner.

In contrast to the EZLGM, the edges of the Ubitrack graph are directed. This is necessary to handle accuracy descriptions of location sensors correctly. As an example, consider a vision-based tracker estimating the 3D position of several ball-shaped locatables. If the arrangement of them is fixed and known, the tracker can compute the orientation of the entire structure. The error of the orientation is then dependent on the error in estimating the balls' positions as well as the size of the baseline used for computations. Consequently, the orientational error can be expected to be higher than the positional error. The situation changes if the camera's pose should be expressed in the object's coordinate system, corresponding to inverting the edge in the SR graph. As can be seen in figure 3.1, now the positional error is high, and the orientational error is low.

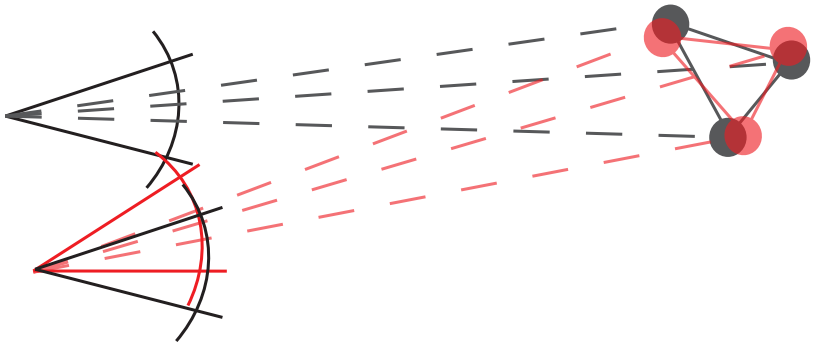


Figure 3.1: Why the SR graph needs to be directed: If the object's real state (black) is measured with some error (red), the positional accuracy remains high, while the orientational accuracy degrades. If we look at the situation from the object's perspective and want to estimate the camera's pose in the object's coordinate system, the positional accuracy degrades, while the orientational accuracy is high.

The graph-based model allows a simple integration of all kinds of sensors, static measurements done by hand or in complex calibration procedures [123] by adding new edges.

3.2.2 Real Relationships

For the sake of clarity, let us first discuss how real relationships, i.e. the "world as God sees it", would be represented in the SR graph model. In this perfect world, every spatial relation between every pair of two objects would be known at any point in time with absolute certainty and without any error. This corresponds to a complete graph, which is of course also transitive, symmetric and reflexive. An example of such a graph is shown in figure 3.2.

To treat the handling of spatial relationship state space in a formal way, we define a binary relation Ω operating on the node set $N = \{A, B, \ldots\}$. Thus, whenever we know anything about the spatial relationship between two objects X and Y (which is always true in the real world SR graph), the relation (X, Y) is true. We then map every element (X, Y) of Ω onto a function w_{XY} by an attribution scheme $\mathbf{W}$:

$$\mathbf{W} : (\Omega = N \times N) \rightarrow w \tag{3.1}$$

Every function w describes the spatial relationship between the objects X and Y in an arbitrary state space $\mathcal{S}$ over time. Thus, w is defined as

$$w : D_t \rightarrow \mathcal{S} \tag{3.2}$$

with D_t being the source time domain, i.e. the set of points in time where we have knowledge about the spatial relationship between X and Y. Again, in the perfect

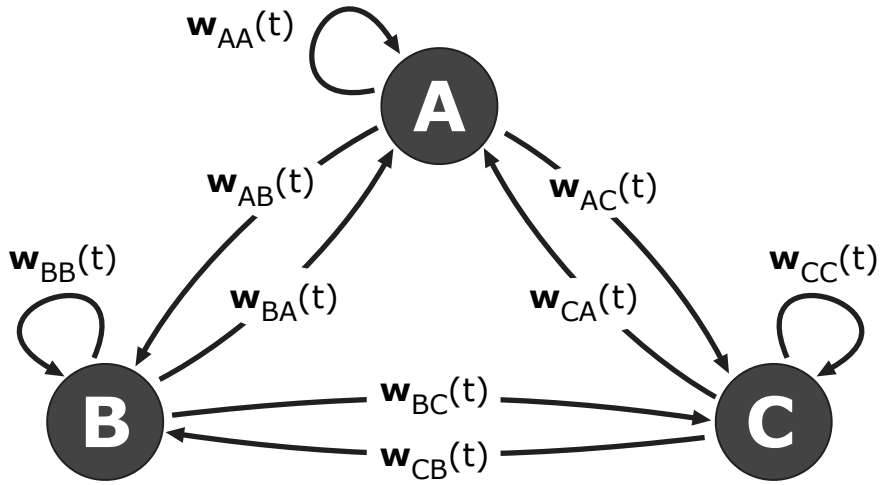


Figure 3.2: A real world SR graph consisting of three objects.

real world graph, D_t is not limited. Examples for the state space $\mathcal{S}$ have been discussed in sections 1.1.3 and 1.2.3. In all existing implementations of the formal model, a 6 DOF pose state space has been chosen, with position represented by a three-dimensional real vector and orientation as a unit quaternion [67]:

$$\mathcal{S}_{6\ DOF} = \{(p_x, p_y, p_z)^T, (o_x, o_y, o_z, o_w)^T\} \tag{3.3}$$

For a clear notation, we define a directed graph $G(\Omega)$ representing the relation Ω. Thus, whenever we know about the spatial relationship of object $Y \in N$ relative to object $X \in N$, the nodes representing these objects are connected by an edge that is annotated with the function w_{XY}.

3.2.3 Measured Relationships

In real tracking setups, it is not possible to obtain a complete spatial relationship graph between all objects. We can only make *measurements* of the spatial state of some *sensors* relative to *locatables* they sense.

The sensors can be of very diverse kinds. The most simple method would be to take a measuring tape and manually determine the spatial relationship of one object relative to another. We can also use a more sophisticated calibration procedure (see [123] for some examples) to get an estimate of the spatial relationship between two objects. Of course, common hardware sensors can obtain repeated measurements and deliver them automatically. Finally, we can write sophisticated software that aggregates the data of several sensors in order to estimate two object's spatial relationships, potentially even predicting the state into the future.

All these possibilities have one thing in common: they are far from perfect, introducing some *error* into the measurement. In section 2.7.3, we discussed several possibilities to model error in estimating spatial state. The Ubitrack formalism must be general enough to allow various error models, not enforcing a particular method on the user. In addition, error is not the only property of a measurement an application using the formalism might be interested in. In consequence, the binary relation Ω and the attribution scheme $\mathbf{W}$ of the real world SR graph must be extended to allow the storage of these properties.

The binary measurement relation Φ operates on the node set N, and an attribution scheme $\mathbf{P}$ maps every element (X, Y) of the relation onto a function p_{XY}:

$$\mathbf{P} : (\Phi = N \times N) \rightarrow p \tag{3.4}$$

In contrast to the real world functions w_{XY}, p_{XY} not only yields the spatial state $\mathcal{S}$, but also a set of *attributes* $\mathcal{A}$:

$$p : D_t \rightarrow \mathcal{S} \times \mathcal{A} \tag{3.5}$$

These attributes store the properties of the individual measurement.

Again, a directed graph $G(\Phi)$ represents the relation Φ. Figure 3.3 shows an example with the node set $N = \{A, B, C\}$. Although the graph may have the same

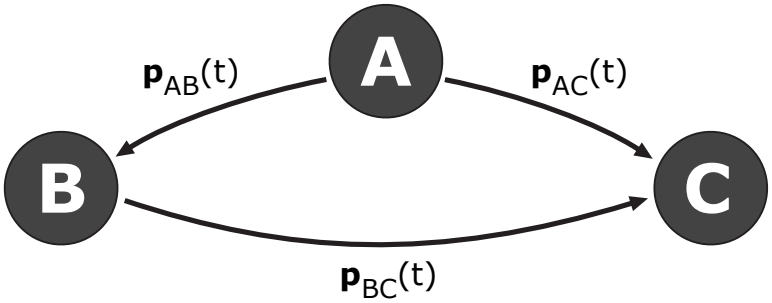


Figure 3.3: Measurement Graph $G(\Phi)$: An edge between two nodes only exists if a measurement of the spatial relationship of the objects they represent has been made.

node set as a real world graph $G(\Omega)$, its edge set is usually much smaller. An edge e_{XY} exists only if at least a single measurement of the spatial relationship between two objects represented by nodes X and Y has been made. In consequence, the graph $G(\Phi)$ is in general *not* transitive, neither is it symmetric.

Timing Issues. A major design goal of the Ubitrack formalism is to handle time in a flexible way, such that both past, present and future estimates of spatial state

can be modeled. For this purpose, we will have a closer look at the definition of the measurement function p_{XY}. This function maps the *source time domain* D_t onto the spatial state and a set of attributes describing the measurement quality. With this definition, we can indeed model all measurements done by a sensor in its whole lifetime with a single function–for every measurement, we add an entry to the discrete finite set D_t.

As an example, assume the "sensor" to be the combination of a measuring tape and an undergraduate student. As part of his AR class, the student has to measure the position and orientation of a poster P in the laboratory in a ceiling-mounted tracker T's coordinate system. He uses the measuring tape and some knowledge from a geometry class, and finally comes up with some value, say $(x, y, z) = (0.12, 0.98, 1.97)$. The only property describing the measurement we can obtain is from some old paper that tells us that students typically make errors of $0.1m$ in every direction when given such a task.

If we assume that the measurements were all done on August 31st, 2003, 11:11 am, we now can add an edge TP to the graph $G(\Phi)$ and a function p_{TP} that is defined as follows:

$$\begin{aligned} p_{TP} \quad &: \quad D_t \to \mathcal{S} \times \mathcal{A} \\ &\quad\;\; t \mapsto p_{TP}(t) \\ &\quad\;\; p_{TP}(31/08/2003,\ 11{:}11\ \text{am}) = \left\{ \begin{pmatrix} 0.12 \\ 0.98 \\ 1.97 \end{pmatrix}, \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \end{pmatrix} \right\} \end{aligned}$$

Unfortunately, the decoration of the lab gets rearranged after some time, therefore another undergrad has to do the measuring again, this time on December 4th, 2003, 12:10 am. The source time domain D_t gets now extended and p_{TP}'s value is

$$p_{TP}(4/12/2003,\ 12{:}10\ \text{am}) = \left\{ \begin{pmatrix} 1.42 \\ 2.03 \\ 0.85 \end{pmatrix}, \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \end{pmatrix} \right\}$$

For this example, the measurement state space $\mathcal{S}$ has been chosen as three degrees of freedom position in meters, and the attribute space $\mathcal{A}$ consists of a simple error model assuming a Gaussian distribution of error with three independent parts describing the variance in x, y and z direction.

If we take electronic sensors such as typical AR trackers, the situation is almost the same–the time domain is discrete and finite. Of course, it has much more elements. For example, a tracker with an update rate of 60Hz yields sixty new elements per second.

3.2.4 Inferred Relationships: Using Spatial Transitivity

The primary goal of the Ubitrack formalism is to provide, at any point in time, an optimal estimate of the spatial relationship between any two objects. The measurement graph $G(\Phi)$ does not fulfill this requirement. Spatial relationships between objects are available only at some very specific points in time and only for a very limited subset of object pairs. As such, we have to *infer* knowledge from the scarce data we are given.

The simplest approach is to use the fact that spatial relationships are transitive. If we have measured the spatial relationship of object B relative to object A and the relationship of object C relative to B, we can aggregate this data and infer the relationship of object C relative to A. However, this can only be done if we have obtained the measurements at exactly the same point in time, a requirement that is almost impossible to fulfill in practice.

In consequence, another step has to be taken before using transitivity: knowledge about spatial relationships has to be inferred for points in time without an existing measurement. Several possibilities exist:

Take last measurement. The standard approach for this problem is to take the last measurement and assume that the real state of the object will not differ too much from this measurement. Obviously, high update rates of trackers make this approach feasible, in fact, it is the most often used solution in existing AR systems.

Simple linear inter-/extrapolation. For some tracking devices, the update rate may not be sufficiently high to allow the "last measurement" approach. However, the underlying movement process may be rather simple, such that a linear inter- or extrapolation becomes feasible. One example application is tracking airplanes via GPS–the usual GPS update rate of 1HZ can be increased without problems.

Complex inter-/extrapolation. Reconsidering the discussion about the Extended Kalman Filter in section 2.7.1, a complex movement model describing an object's state space can be employed to make a more accurate inference.

All those possibilities introduce additional errors. Analogously to errors and other properties describing measurements, we can define an attribute space $\mathcal{A}$ for describing inferences, and extend our graph-based model for inferred relationships.

The binary inference relation Ψ operates on the node set N, and an attribution scheme $\mathbf{Q}$ maps every element (X, Y) of the relation onto a function q_{XY}:

$$\mathbf{Q} : (\Psi = N \times N) \rightarrow q \tag{3.6}$$

As with the measurement function p_{XY}, the inference function q_{XY} yields a spatial state $\mathcal{S}$ and a set of attributes $\mathcal{A}$:

$$q : D_t \rightarrow \mathcal{S} \times \mathcal{A} \tag{3.7}$$

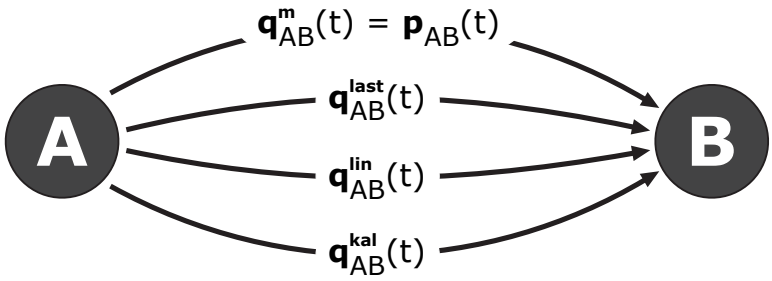


Figure 3.4: Inference Graph $G(\Psi)$: An individual edge is added for every inference of the spatial relationship between objects represented by two nodes.

A directed graph $G(\Psi)$ represents the relation Ψ. Figure 3.4 shows an example graph with two nodes. For every measurement or inference function describing the spatial relationship between two objects represented by the nodes X and Y, an edge e_{XY} gets added to the graph. In consequence, there may be multiple edges between two nodes, each associated with a different measurement or inference function. In the example, $q^m_{AB} = p_{AB}$ is the original measurement function, q^{last}_{AB} describes the inference "take last measurement", q^{lin}_{AB} the linear interpolator and q^{kal}_{AB} the Kalman filter approach.

Example: Tracked object calibration. For a more complex example, consider how to model a calibration component. In this standard situation for every AR setup, we have a world coordinate system W and a tracker coordinate system T. For the sake of simplicity, we assume both to be identical. All spatial relationships are in a 6 DOF pose state space, i.e. they can be expressed by a 3D position vector and a quaternion for the orientation. The tracker detects the spatial state L of a locatable that is rigidly attached to some object. The object's coordinate system O is not identical to L. In consequence, some calibration algorithm, such as matching known points in coordinate systems O and L, has to be applied. Let us further assume that the system's user has already performed the calibration procedure and has the transformation from L to O. The corresponding spatial relationship graph is shown in figure 3.5, containing two measurement functions $q^m_{TL}(t) = p_{TL}(t)$ and $q^m_{LO}(t) = p_{LO}(t)$. Note that $q^m_{TL}(t)$ is only defined at all those points in time when the tracker delivers a measurement and $q^m_{LO}(t)$ is only defined at a single point in time t_c when the calibration has been done.

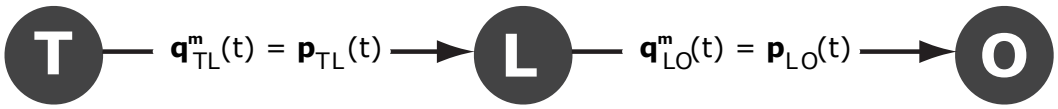


Figure 3.5: SR graph of simple AR tracking setup.

A software component that permanently takes the last available value of $q^m_{TL}(t)$ and multiplies it with the static value $q^m_{LO}(t)$ can be modeled as follows:

1. Create an inference function $q^e_{TL}(t)$ that is defined on a continuous time domain and yields the last available value of $q^m_{TL}(t)$. This leads to a new edge between nodes T and L in the graph.

2. Create an inference function $q^e_{LO}(t)$ that is defined on a continuous time domain and yields the static calibration parameter $q^m_{LO}(t_c)$ for all input values t.

3. Define an inference function $q^i_{TO}(t)$ that takes the values of $q^e_{TL}(t)$ and $q^e_{LO}(t)$ as input values and yields the transitive spatial relationship between T and O by multiplying the spatial relationships between T and L and L and O.

Figure 3.6 shows the resulting SR graph, with newly inferred edges drawn dashed and in green color.

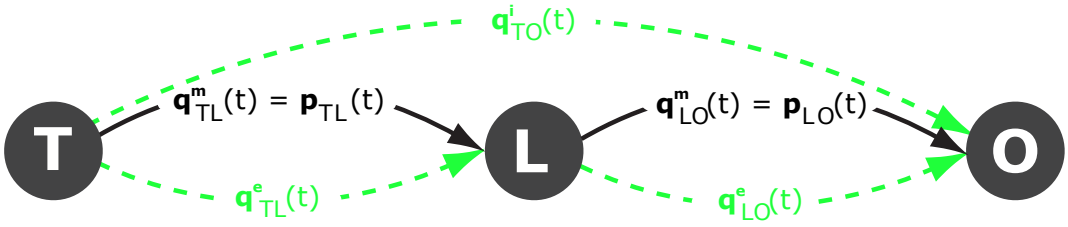


Figure 3.6: Inferred SR Graph of simple AR setup.

Propagation of spatial relationships for 6 DOF pose state space. If we just consider the spatial state space $\mathcal{S}_{6\ DOF} = \{(p_x, p_y, p_z)^T, (o_x, o_y, o_z, o_w)^T\}$, we can derive a generic inference function $q^i_{AC}(t)$ that yields the spatial relationship of

object C relative to object A, given functions $q_{AB}(t)$ and $q_{BC}(t)$. The position $p_{AC}(t)$ is computed as

$$p_{AC}(t) = p_{AB}(t) + o_{AB}(t) \cdot p_{BC}(t) \cdot o^*_{AB}(t) \tag{3.8}$$

and the orientation $o_{AC}(t)$ (given as quaternion) as

$$o_{AC}(t) = o_{AB}(t) \cdot o_{BC}(t) \tag{3.9}$$

with "$\cdot$" being the quaternion multiplication.

3.2.5 Evaluation Function

Spatial relationships can thus be aggregated along a path in the spatial relationship graph $G(\Psi)$. One question remains: if there are multiple paths between object X and Y, which one should be chosen to satisfy an application's needs?

Definition. To cope with this problem, the Ubitrack formalism introduces an application-supplied *evaluation function* e. Its purpose is to provide a quality estimate of an inference made along a specific path. It maps the attributes of a path onto a real number:

$$e \;:\; \mathcal{A}^* \rightarrow \mathbb{R} \tag{3.10}$$

$$\mathcal{A}^*_i \mapsto e(\mathcal{A}^*_i) \tag{3.11}$$

By convention, a smaller value of $e(\mathcal{A}^*_i)$ signifies a better fulfillment of the application's needs. The general definition of the evaluation function provides a plug-in mechanism for designing arbitrary optimization criteria; it is the responsibility of the designer of the evaluation function to select a preferred evaluation criteria. To favor, for example, low-latency inferences over those with low absolute spatial errors, the overall function could be the weighted sum of two normalized functions expressing latency and spatial error. The latency function's weight would then be higher than the error function's.

Rationale. The plug-in mechanism for the evaluation function, along with an attribute set that is open to new extensions, ensures that the Ubitrack formalism is applicable to yet unknown application domains. New application areas can make use of an already existing Ubitrack model of a sensor network by providing evaluation functions that work on existing attribute sets or by extending the attribute set.

In addition, the evaluation function concept allows to serve multiple application with differing needs simultaneously. The Ubitrack formalism is used to model all knowledge about available measurements and inferences in an abstract way. Using

the evaluation function, each application accessing the information stored in the model can specify individually how to weigh the properties of spatial relationship estimates.

Evaluation Function Examples. To make the concept of evaluation functions more clear, let us discuss several real-world examples. For all these, we assume an attribute set $\mathcal{A}$ consisting of the following values:

- A latency value l given in milliseconds;
- an update rate u given in Hz;
- a confidence value between 0 and 1; and
- a 6×6 covariance matrix according to the error model described in section 2.7.3.

Assume a 3D gaming application. The most important aspect for games is low latency and a high update rate to ensure the user's immersion into the virtual world; accuracy is not as important. In consequence, we might choose an evaluation function that ignores the error and the confidence value, but tries to make a tradeoff between latency and update rate:

$$e_{\text{game}} = l + \lambda u^{-1}$$

with λ being a weighing value that has to be adjusted properly.

The situation is different for a medical AR application, where accuracy is most important. It is more acceptable that the doctor is forced to move slowly or uses stationary pre-arranged viewing setups than having an error in e.g. computer tomography data projected onto a patient's body for minimally invasive surgery. Let us further assume that the application projects small dots onto the body, indicating the places where the doctor has to make a cut. The orientational accuracy of these dots is not very important, but the positional accuracy is. In consequence, we design an evaluation function that almost exclusively considers the overall positional accuracy.

A more complex evaluation function is necessary to make use of the error model discussed in section 2.7.3. Error is represented as a 6×6 covariance matrix $\mathbf{C}$. The trace of a matrix is the sum of its eigenvalues, and the eigenvalues are the square lengths of the axes of the confidence ellipsoid for $\sigma = 1$. Thus, a smaller trace corresponds to a smaller overall Gaussian error. It is advisable to apply a weighting matrix $\mathbf{A}$ to the covariance $\mathbf{C}$, such that specific errors can be weighted more or less. As an example, for visual augmentations the depth information is not as important as the directions perpendicular to the viewing direction.

In consequence, the evaluation function for the error model used throughout this thesis is

$$e_{\text{Ubitrack}} = \sqrt{\text{trace}\left(\mathbf{ACA^T}\right)} \tag{3.12}$$

3.2.6 Attributes Describing Measurements and Inferences

In the examples above, we have only mentioned how to propagate the spatial state when making use of the transitivity of spatial relationships, whereas the propagation rules for attributes have been ignored.

In this section, possible attribute sets will be presented. The choice was motivated by the analysis of existing tracking devices in section 2.5 and is not exhaustive. Other attribute sets might be added to the formalism as needed, as long as some propagation rules can be provided.

For all these attribute sets, propagation rules are given that can be used to define generic inference functions. These rules can infer not only the spatial relationship between two objects X and Y, but also the attribute set.

The rules assume that we are looking for a particular attribute $a(P)$ of a path P consisting of the nodes

$$N_1, N2, \ldots, N_{n+1}$$

and consequently the edges

$$e_1 = e_{N_1N_2}, e_2 = e_{N_2N_3}, \ldots, e_n = e_{N_nN_{n+1}}.$$

They all take the attributes $a(e_1), a(e_2), \ldots, a(e_n)$ as input.

Latency. This attribute is one of the most important ones for typical AR applications. It is usually given in seconds, and gives the time delay between an object's state in the real world and the time when information about this state is available to the computing system. In consequence, if we have latency l for a spatial relationship q_{XY}, knowledge about the value of $q_{XY}(t_0)$ is only available at times $t \geq t_0 + l$. Latency is sometimes referred to as "lag". Note that using prediction components in a tracking setup can reduce the numeric value of the latency attribute, it may even become negative, for the price of decreasing spatial accuracy.

The propagation rule is based on the observation that in a chain of inferences the element with the largest latency dominates all others. Thus, we take the maximum of all latencies along the path:

$$l(P) = \max_{i=1}^{n} l(e_i) \tag{3.13}$$

Update rate. The *measurement frequency* or *update rate* of a sensor indicates how often new measurements become available.

The value of the update rate can differ widely, ranging from several Kilohertz for fast accelerometers or gyroscopes down to almost zero for static measurements done only once in a setup's lifetime, such as the internal calibration parameters of a tracker.

It is very hard to find an accurate propagation rule for the update rate, since the usually unsynchronized update rates of the functions to be aggregated lead to a complex behavior of the overall update rate.

Whenever some spatial relationship along the aggregated path has been updated, the aggregated relationship can be regarded as updated as well, leading to an average update rate that is the maximum of all involved rates, i.e.:

$$u(P) = \max_{i=1}^{n} u(e_i) \tag{3.14}$$

On the other hand, one could argue that it is better to think of the inverse of the update rate, i.e. the time between subsequent updates. We can only be sure that the aggregated spatial relationship is updated whenever the least common multiple (LCM) of all update times has passed:

$$u(P) = \frac{1}{LCM(u(e_1)^{-1}, u(e_2)^{-1}, \ldots, u(e_n)^{-1})} \tag{3.15}$$

Which propagation rule should be chosen depends on the individual application domain of the formalism and can not be generalized.

Confidence. Many tracking devices employ pattern matching. For example, vision based tracking devices usually work by matching a camera image to some patterns, ultrasonic trackers work by matching an observed sound signal to some predefined pattern. This matching process can be erroneous, leading to misclassifications. In consequence, a sensor might yield a value that does not correspond to the true state of an object. To reflect this fact, a *confidence value* can be used as an attribute. It is a real number in the interval $[0; 1]$, indicating the probability that the given spatial relationship is valid.

For a generic propagation rule, we have to assume that the individual measurements are stochastically independent:

$$c(P) = \prod_{i=1}^{n} c(e_i) \tag{3.16}$$

Of course, this assumption often does not hold. Especially vision-based trackers often suffer e.g. from bad lighting conditions simultaneously. However, if the manufacturer of such a device can tell how the confidence values depend upon each other stochastically, we can reflect this knowledge in a modified propagation rule.

Monetary Cost. The price you pay for a measurement will become more and more important if tracking applications pervade the everyday life. Although today's AR and ubicomp setups usually do not care about this attribute, as the fixed cost for the sensor infrastructure is very much higher than the cost per measurement. Even today estimates of spatial relationships can be bought as a special service of some cell phone providers. Based on signal strength of a phone at several base stations, a phone's owner can obtain its rough position via a web page or SMS (Short Message Service), and usually has to pay less than 1 EUR per request.

The propagation rule consists of adding up the cost for all measurements involved:

$$mc(P) = \sum_{i=1}^{n} mc(e_i) \tag{3.17}$$

Measurement Error. Measurement error is by far the most interesting and complex attribute. It indicates by some metric how accurately the estimate of a spatial relationship between two objects reflects the true state. In section 2.7.3 some possibilities have been discussed, here some issues arising when choosing a measurement error model as attribute in the formal Ubitrack model are discussed.

First, an error model is usually only adequate for a specific set of dimensions of a measurement state space, i.e. if attributes corresponding to measurement error need to be propagated along a transitivity path, all edges in the SR graph belonging to this path have to express the same dimensions of the same sensor state space.

Second, care has to be taken that the computational cost of an error model is kept below reasonable limits, especially if propagation of error has to be estimated. For some applications it may be necessary to have constant estimates of the overall error [75], thus the propagation has to be computed with every update of a spatial relationship.

Third, it may be reasonable to use different error models for different sensors, as the characteristics of the underlying error distributions may differ significantly. Whilst a Gaussian distribution is a suitable model for many dynamic trackers that are typically used in AR applications, the most suitable error distribution to describe static measurements, such as a table's 6 DOF pose in a room, differs completely: for a very long time, the estimate is only disturbed by very small Gaussian noise resulting from slight movements of the overall setup, but as soon as someone hits on the table, a large error occurs. The problem resulting for the Ubitrack formalism is how to aggregate such diverse error models such that a correct propagation of attributes can be assured.

3.2.7 A generic algorithm for finding optimal inferences.

Spatial transitivity in combination with an evaluation function gives us a generic algorithm for finding an optimal inference of the spatial relationship between two objects X and Y:

Algorithm 1: Generic algorithm for finding optimal inferences

Data: Source node X, target node Y, time T, evaluation function e
Result: Optimal transitive estimate of spatial relationship between X and Y.
begin
  Find all paths from X to Y in graph $G(\Psi)$, with the additional constraint that an edge corresponding to an inference function $q(t)$ is only considered if q is defined at $t = T$. ;
  Evaluate the function e over all the detected paths. ;
  Use the path corresponding to the lowest value of e to calculate the inferred spatial relationship q^i_{XY} between X and Y. In addition, compute the attribute set of this inference. ;
  Add an edge e_{XY} to $G(\Psi)$, with q^i_{XY} being the associated inference function. ;
end

Note that we have to be careful when adding an edge to the graph that has an associated function that depends on other functions' attributes: If these attributes change at a later point in time, the new inference function has to reflect these changes. This has to be reflected both in a formal definition and a runtime implementation of the inference function.

Example. A small example with three nodes illustrates how the algorithm works. Suppose we are given three objects, A, B and C. Measurements are provided by some trackers, leading to functions p_{AB}, p_{AC} and p_{BC}. We assume the state space to be 3 DOF position, and the attribute set to consist solely of the measurement latency, measured in milliseconds. We further assume that the timing issues have been resolved using the "take last measurement" strategy, leading to functions q^e_{AB}, q^e_{AC} and q^e_{BC}.

An application is now interested in the spatial relationship of object C relative to object A, with the additional requirement of having minimal latency in the overall estimate. In consequence, the evaluation function consists of the latency. Remember that a lower value of the evaluation function indicates better suitability for the application.

First, the algorithm finds all paths from A to C:

$$\mathbf{P1} : A \xrightarrow{q^e_{AB}} B \xrightarrow{q^e_{BC}} C, \qquad \mathbf{P2} : A \xrightarrow{q^e_{AC}} C$$

We assume the application has made a request for a time when no direct measurement is available, in consequence all paths involving functions $q^m = p$ are excluded.

Second, it evaluates e over these paths:

$$\begin{aligned} e(\mathbf{P1}) &= \max(2,5) = 5 \\ e(\mathbf{P2}) &= \max(10) = 10 \end{aligned}$$

Third, it computes the inferred spatial relationship along the optimal path **P1**, in our case a 3 DOF position, by adding the position values coming from q^e_{AB} and q^e_{BC}, and the inferred attribute set according to the propagation rules, leading to a inferred latency of 5ms.

Fourth, a new edge e_{AC} with a corresponding inference function q^i_{AC} gets added to the graph $G(\Psi)$. Figure 3.7 shows the updated graph.

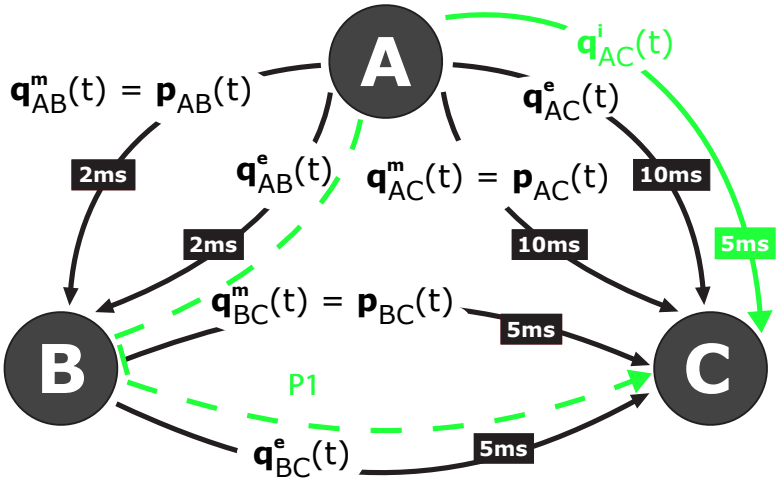


Figure 3.7: Example of generic inference algorithm.

3.2.8 Pathwise and Edgewise Evaluation Functions

Up to now, we defined the evaluation function to map a whole path in the SR graph onto a real positive number. This general class of evaluation functions is called *pathwise evaluation function* in the Ubitrack formalism. The generic algorithm just discussed has to find all paths between two nodes to select an optimal solution. Unfortunately, the number of paths between two arbitrary nodes is exponential in the number of nodes of a graph, in consequence, the algorithm is of little use in practice.

In contrast, *edgewise evaluation functions* operate on single edges. If a whole path needs to be evaluated, we take the sum of the evaluation function values along the edges in the path, i.e.:

$$\begin{aligned} e' : \mathcal{A} &\rightarrow \mathbb{R} && (3.18)\\ \mathcal{A}_i &\mapsto e'(\mathcal{A}_i) && (3.19)\\ e(p) &= \sum_{i=1}^{n} e'(N_{i-1}N_i) && (3.20) \end{aligned}$$

with p being the path consisting of the nodes $N_1 N_2 \ldots N_{n-1}$. If the evaluation function is structured like that, we can assign every edge a weight based on its evaluation function's value. Then, a standard shortest path algorithm to compute an optimal path from N_1 to N_{n-1} can be applied.

Shortest path algorithms are a well researched topic. In general, two classes exist, label-setting and label-correcting algorithms [4].

Single-Source Shortest Path Problems. The complexity of determining the shortest path between two nodes is the same as determining the shortest path from a single source node to all other nodes. Consequently, this problem is also known as *Single Source Shortest Path (SSSP)*.

Label-setting shortest path algorithms: Label-setting algorithms work in an iterative fashion over the node set and designate a permanent label to a single node at each iteration.

For this purpose, they keep two sets of nodes: a set consisting of nodes with *permanent* labels and a set with *temporary* labels. A permanent label is only assigned once to each node and gives the shortest distance from this node to the source node. Temporary labels may be assigned multiple times and give an upper bound on the shortest distance to the source node. Initially, the source node gets assigned the temporary label 0 and all other nodes get the temporary label $+\infty$. At every iteration, a node with minimum temporary label is assigned this label permanently. Its neighboring temporary nodes' labels are updated according to the weights of the edges between them and the new permanent node. Thus, at the ith step, the algorithm has found all shortest paths consisting of a maximum of i nodes. It terminates if all nodes are assigned permanent labels.

The variants of label-setting algorithms differ in how the set of temporary nodes is represented as a data structure. Dijkstra's algorithm [36] is the most prominent representant of the label-setting class. Other algorithms build upon Dijkstra's ideas and differ in their data structures. For dense networks, Dijkstra's original implementation achieves the best running time, $\mathcal{O}(n^2)$ with

Algorithm 2: Label-setting single source shortest path

```
begin
    P ← ∅ set of permanent nodes ;
    T ← N set of temporary nodes ;
    d(i) ← ∞ for each node i ∈ N ;
    while |P| < |N| do
        select a node i ∈ T for which d(i) = min{d(j) : j ∈ T} ;
        P ← P ∪ {i} ;
        T ← T − {i} ;
        foreach (i, j) ∈ A(i) do
            if d(j) > d(i) then d(j) ← d(i) + c_ij and pred(j) ← i ;
end
```

n being the number of nodes. In sparse networks (as typically the SR graph will be), a rather complex Fibonacci heap implementation achieves a better running time of $\mathcal{O}(m + n \log n)$. For a detailed discussion, see [4].

Label-setting algorithms are only applicable to shortest path problems on either noncyclic directed graphs with arbitrary edge weights or arbitrary graphs with nonnegative edge weights, as the shortest path problem gets undefined with negative cycles in the graph. Label-setting algorithms do not detect this problem and will not terminate then. The SR graph of the Ubitrack formalism is directed, but cycles might well occur and negative edge weights are allowed, too. However, reconsidering the discussion on attributes in section 3.2.6, an evaluation function can almost always be designed in a way that makes it strictly nonnegative. A notable exception are evaluation functions based on a latency attribute set, as predictive components have a negative latency attribute.

Label-correcting shortest path algorithms: Label-correcting algorithms are iterative as well. All are variants of the generic algorithm. It maintains a set of distance labels d at every stage. A label $d(i)$ is either ∞ if no path from the source node to i has yet been found or it is the length of some directed path from the source node to i. In addition, a predecessor index $pred(i)$ is kept that tells us the predecessor of node i on the current path from the source node to i. At every step, the algorithm selects a node whose weight can be reduced and updates its label accordingly. At termination, no such node exists.

Label-correcting algorithms can also detect negative cycles in directed graphs and can therefore be applied directly to every SR graph and evaluation function.

The generic label-correcting algorithm was proposed by Ford and later modi-

Algorithm 3: Label-correcting single source shortest path

```
begin
    d(s) ← 0 and pred(s) ← 0 ;
    foreach j ∈ N − {s} do d(j) ← ∞ ;
    while some arc (i, j) satisfies d(j) > d(i) + c_ij do
        d(j) ← d(i) + c_ij ;
        pred(j) ← i ;
end
```

fied by Bellman. He proposed to use a FIFO list for selecting edges leading to nodes whose labels should be corrected. At every step, all shortest paths up to an increasing length have been discovered. The Bellman-Ford algorithm has the currently best strongly polynomial running time for label-correcting algorithms, $\mathcal{O}(nm)$.

All Pairs Shortest Path Algorithms. Sometimes it might be reasonable to precompute the shortest paths between all pairs of nodes in a graph. Two major algorithms exist for the *All Pairs Shortest Path (APSP)* problem.

First, a label-setting algorithm can be applied repeatedly to all nodes, leading to a running time of $\mathcal{O}(n^3)$ for Dijkstra's algorithm and $\mathcal{O}(nm + n^2 \log n)$ for the Fibonacci heap implementation. Still, no negative cycles are allowed in the graph.

Floyd and Warshall developed a label-correcting algorithm based on dynamic programming that achieves a running time of $\mathcal{O}(n^3)$ and can detect negative cycles in the graph.

3.2.9 Complex Inferences

The shortest path based inference algorithm just described has the advantage of being generic, such that we can make use of it in every location sensor network without the need to have extra knowledge about the physical properties of the objects we track.

However, as has been shown in section 2.8, a lot of research has been put into using world knowledge to make sensor fusion work better. The Ubitrack formalism was designed such that these existing efforts can be integrated seamlessly, thereby combining the strengths of previous approaches with the benefits of the general Ubitrack approach.

Two classes of complex inferences exist: either external world knowledge is used to increase the quality of a single sensor's readings or multiple sensors measuring the same spatial relationship are aggregated.

Using External World Knowledge. The most prominent approach of this class is using a Kalman filter and a suitable movement model to filter noise out of sensor measurements. Configuring the Kalman filter is a tedious task and has to incorporate situation-specific knowledge. However, once someone has set up such a filter, it can be added easily to the Ubitrack formalism. Assume that the spatial relationship between a sensor S and a locatable L is measured and yields a measurement function q^m_{SL}. The Kalman filter takes this function and some parameter vector par to recursively estimate the spatial state with less noise. In consequence, we can describe the filter as a function

$$q^k_{SL}(t) = f(q^m_{SL}(t), par).$$

If we now add an edge e_{SL} to the inference SR graph that corresponds to $q^k_{SL}(t)$, we can make full use of the Kalman filter. Figure 3.8 illustrates the resulting graph.

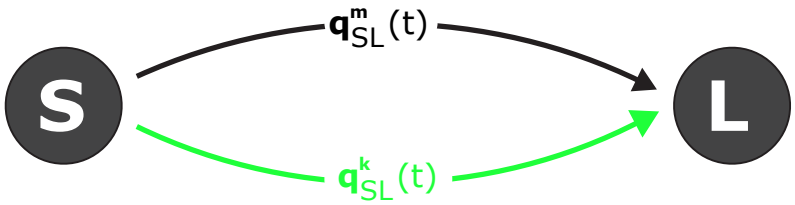


Figure 3.8: Modeling a Kalman filter in the Ubitrack formalism.

Fusing Multiple Sensors. A standard way of sensor fusion consists of taking two sensors with complementary characteristics and let them estimate the spatial relationship of the same locatable relative to a common coordinate frame. If the characteristics of both sensors are known, overall errors can be minimized and the resulting estimate of the locatable's spatial relationship relative to a common coordinate frame is of much higher quality.

The example depicted in figure 3.9 shows how to model such a setup. Two sensors, S_1 and S_2 estimate the spatial relationship of a single locatable L, leading to functions $q^m_{S_1L}(t)$ and $q^m_{S_2L}(t)$. The static relationship between both sensors relative to a world coordinate system is expressed by the functions $q^m_{WS_1}(t)$ and $q^m_{WS_2}(t)$. The sensor fusion can now be expressed as an inference function $q^f_{WL}(t)$ that takes $q^m_{S_1L}(t)$, $q^m_{S_2L}(t)$, $q^m_{WS_1}(t)$ and $q^m_{WS_2}(t)$ as arguments. This function is represented by a new edge from node W to node L in the SR graph.

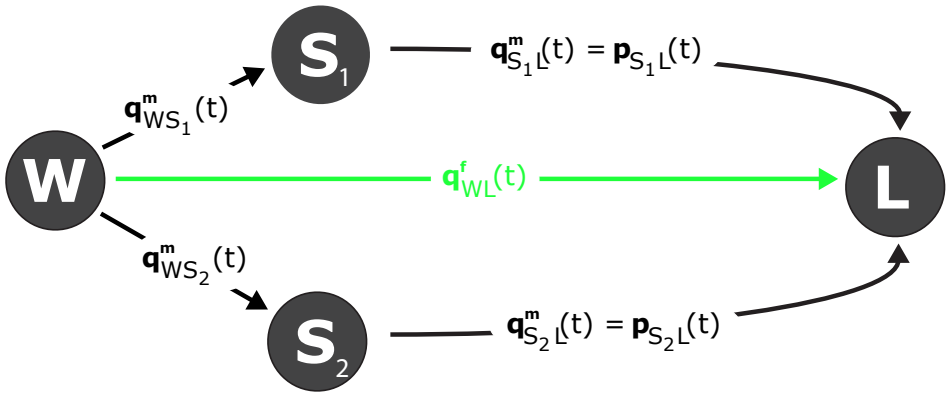


Figure 3.9: Modeling complementary sensor fusion in the Ubitrack formalism.

3.3 Examples

The Ubitrack formalism not only serves as the theoretical basis for the tracking abstraction middleware discussed in the next chapters, but can also be used as a convenient notation facility to describe existing tracking setups in a unified fashion. This section describes several of these, both to show the flexibility of the Ubitrack formalism and to give the reader an intuition of how to apply the formalism to real world problems.

3.3.1 Extending Tracker Ranges

All trackers have a limited range of operation, for example, vision-based trackers need the line of sight. In this example, this problem is remedied by employing an existing tracker to measure the spatial state of a mobile video camera that in turn tracks a locatable beyond the range of operation of the first tracker. In our setup, an ART tracker detects the pose of a video camera that feeds an ARToolkit based tracker. The inferred knowledge of both trackers can then be used to extend the ART system's range. The setup depicted in figure 3.10 shows a simple example application. A projector is used to display a virtual sheep onto the marker, which can not be detected by the ART system.

To facilitate the overall computation, we assume that the projector coordinate system is defined by its projection surface. In addition, we calibrate the ART coordinate system such that it is identical to the projector's. This can be expressed in the SR graph by two edges between the nodes A and P representing the ART system and the projector, both being associated with the identity transformation q^{ident}. The ART system measures the spatial relationship of the locatable L, yielding a function q^e_{AL}. The ARToolkit yields the function q^e_{CM}. Finally, we have measured the offset from the ART locatable L to the camera's center of projection C, yielding

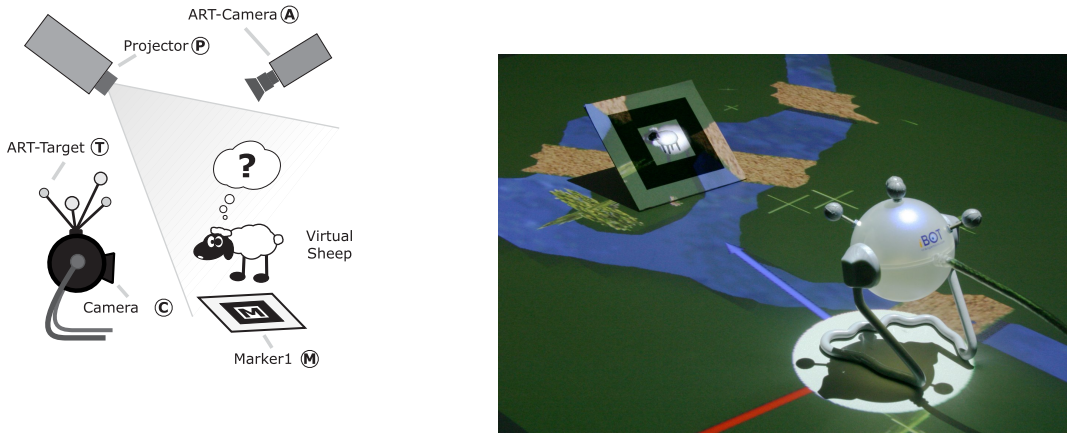


Figure 3.10: Extended tracker range setup: An ART system tracks a camera that detects a fiducial marker outside the ART system's range. This information gets used to project a virtual sheep onto the marker.

a constant function q^e_{LC}.

The sheep projecting application requires the spatial relationship of the marker M relative to the projector P, thus we have to infer knowledge along the path

$$P \xrightarrow{q^{\text{ident}}} A \xrightarrow{q^e_{AL}} L \xrightarrow{q^e_{LC}} C \xrightarrow{q^e_{CM}} M$$

leading to a new edge e_{PM} in the graph with a corresponding inference function q^{app}_{PM}. The resulting graph is depicted in figure 3.11.

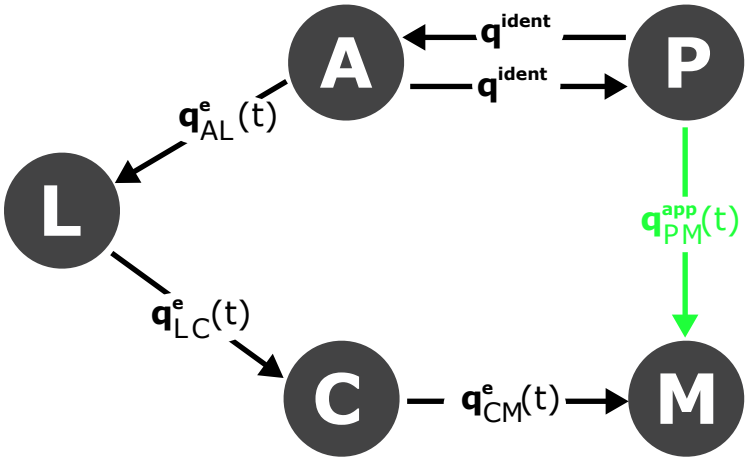


Figure 3.11: Extended tracker range graph: The data along a directed path from P to M is aggregated to create a new inference.

3.3.2 Shared Optical Tracking

Dynamically Shared Optical Tracking [70] is a tracking application based on OpenTracker [99] that handles occlusions within ARToolkit based vision based trackers by using two cameras with partially overlapping viewing frustum. The application assumes two mobile users, both equipped with a head-mounted camera and their own AR system. If both cameras detect a common marker, and one of these detects another marker as well, it can transfer this marker's pose to the second user's system via a wireless network.

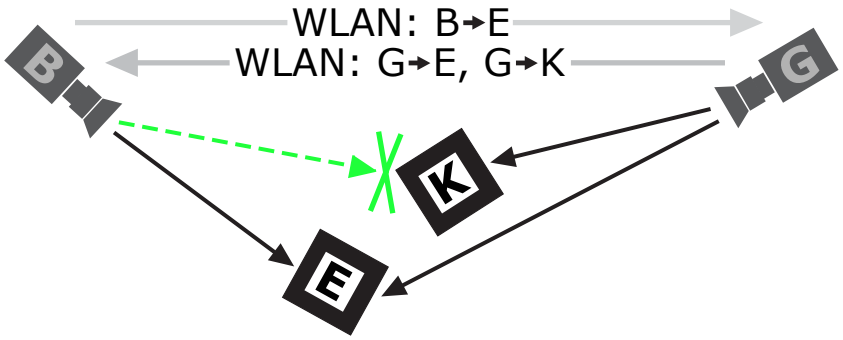


Figure 3.12: Shared optical tracking setup: Two cameras try to detect the pose of two markers. Missing information is exchanged over a wireless network.

Figure 3.12 shows an example setup with two users, B and G, both carrying a camera on their heads. These cameras track two markers, E and K, that subsequently get augmented with some virtual objects in the video image. In the scenario shown in the figure, camera B can not detect marker K's pose, therefore G's AR system transmits its knowledge to B's system that can then compute K's pose relative to camera B.

Figure 3.13 shows the corresponding SR graph of the Ubitrack formalism. For the sake of simplicity, the original measurement functions q^m are not shown. Instead only the extended time domain functions q^e. B's application needs the spatial relationship of marker K relative to camera B. As such, a path from node B to node K has to be found. For this purpose, we first have to invert the function q^e_{GE}, leading to a new edge e_{EG} with a corresponding function q^{inv}_{EG}. This edge will now be part of the path

$$B \xrightarrow{q^e_{BE}} E \xrightarrow{q^{inv}_{EG}} G \xrightarrow{q^e_{GK}} K$$

that is detected by the generic inference algorithm to infer the spatial relationship of K relative to B as requested by the application. This inference is expressed by a new edge e_{BK} and a corresponding function q^{app}_{BK}.

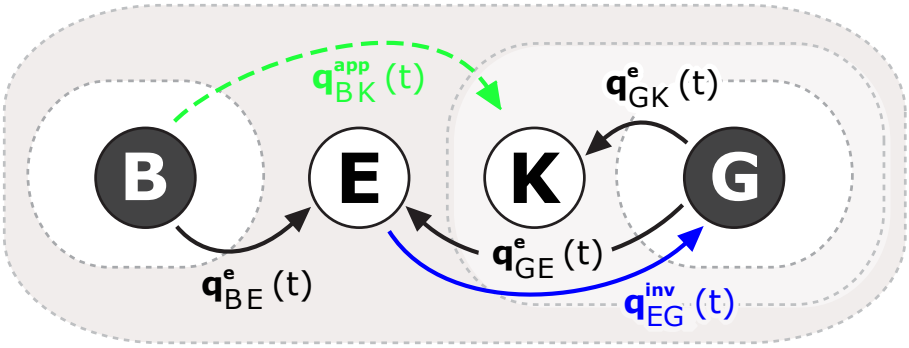


Figure 3.13: Shared optical tracking SR graph: Edge e_{GE} needs to be inverted, then a path from B to K is found and used to infer knowledge.

3.3.3 SHEEP

The *Shared Environment Entertainment Pasture (SHEEP)* [78] is a multiplayer shepherding game with tangible and virtual sheep in a pastoral landscape. The landscape is projected onto a table from above (figure 3.14).

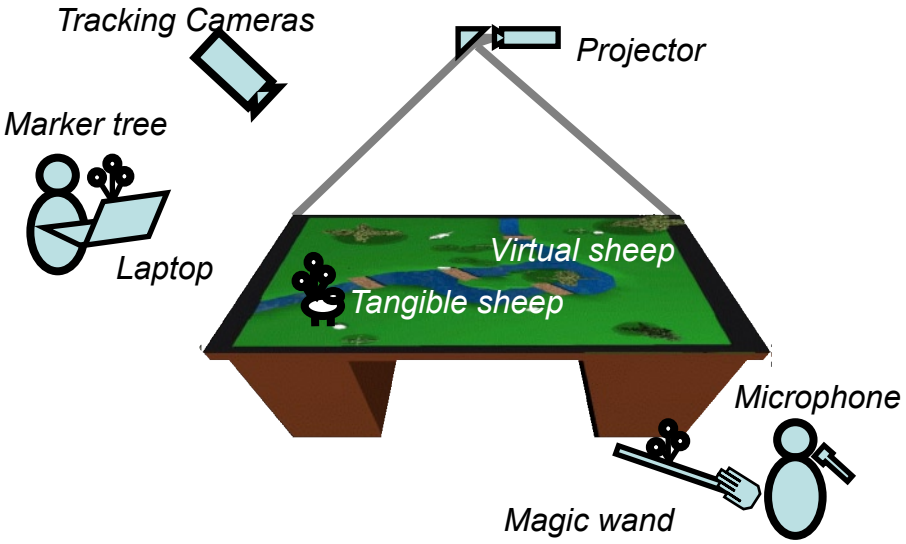


Figure 3.14: The SHEEP game setup.

The application focusses on interaction techniques for AR. On the table, a herd of virtual sheep roams around the pasture. Their behavior can be influenced by a tangible, "real" sheep on the table. Users can insert, delete, scoop and drop sheep with a tangible user interface. Spectators can see a three-dimensional view

of the landscape on the screen of a laptop that can be freely moved about. The sheep appear in their correct three-dimensional position. In addition, a special view through the windscreen of a tracked car is available [15]. The car's current speed is indicated by its speedometer, in addition, an arrow in the car's navigation system constantly points at the next sheep around. Figure 3.15 shows the game in action.

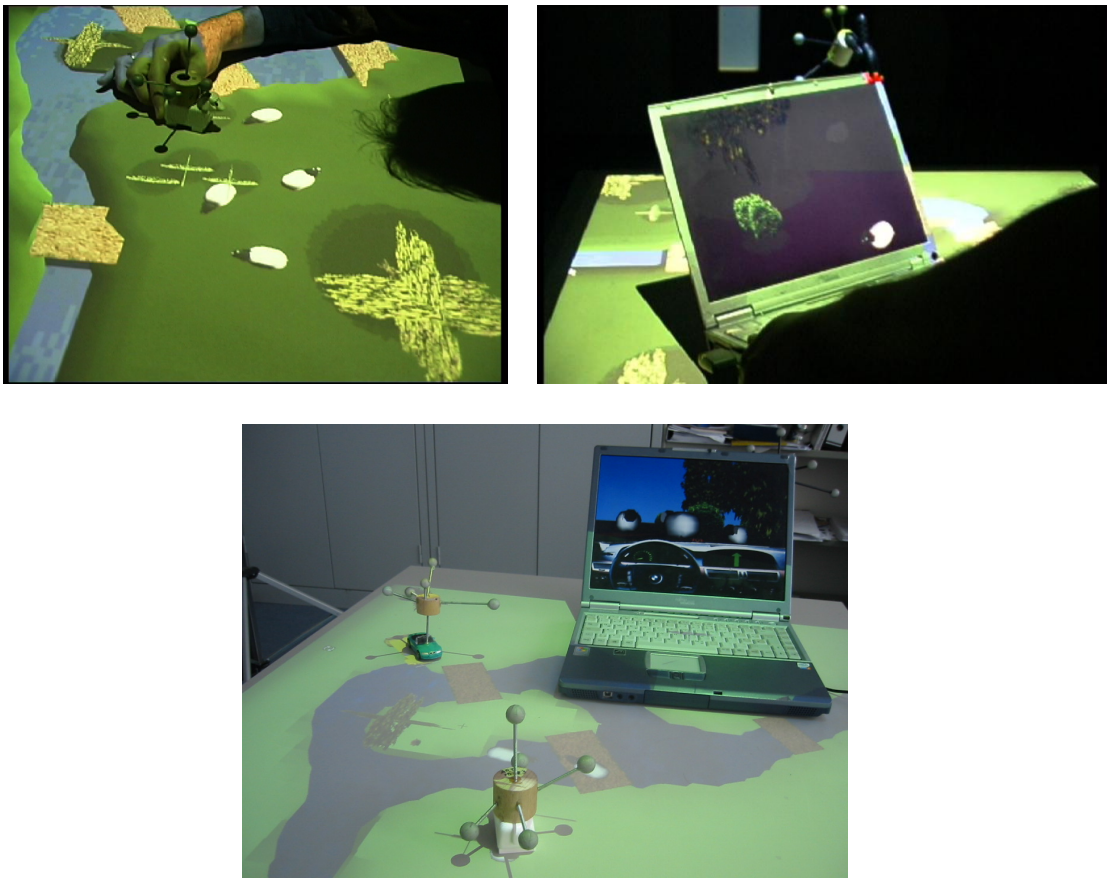


Figure 3.15: Pictures of the SHEEP game. On the top left, a player moves the tangible sheep, and the herd of virtual sheep follows it. On the top right, a spectator uses a laptop to view the scenery in three dimensions. The bottom image shows the view through a tracked car's windscreen.

In the original version of sheep, the tracking setup was simplistic and inflexible. It consisted of an ART optical tracking system that tracked several "marker tree" locatables mounted. The ART system was calibrated in a way that made the projection table the center of the coordinate system. As soon as the table was moved accidentally, the ART system had to be recalibrated. The relationship between the ART locatables and the real objects they were attached to was determined by a so-called *ObjectCalibration* software that used a pointing device for an interactive calibration procedure. The relationship of the pointing device's tip and its locatable

was determined once in an offline procedure and hardcoded into the software. All these inherent assumptions in the code made it extremely difficult to modify the tracking setup to e.g. a new tracking hardware.

Using the Ubitrack formalism provides an efficient way to express all spatial relationships in the demo in a way that makes it easy to extend or modify it later on. Figure 3.16 shows the SR graph of the SHEEP setup. There are several interesting points in this real world model:

- The speedometer indicates the current speed of the car. This can be modeled by an additional edge e_{TC} with an associated function that yields a one-dimensional velocity state space q^v_{TC}. This function takes the 6 DOF pose expressed by q^e_{TC}, takes the three positional values, makes a derivative of these and finally takes the norm of the 3D derivative vector. The pose of the speedometer's hand's angle relative to the dashboard is derived directly from q^v_{TC}'s value.

- The *ObjectCalibration* software works in two steps: first, the pointing device gets calibrated, i.e. the transformation from the magic wand to its tip is determined [123]. The software subsequently uses this information to calibrate the other objects, the car, the tangible sheep and the laptop. In addition, the relationship between the projection table and the ART system can be calibrated, leading to a new edge e_{TA}. The *ObjectCalibration* software not only provides the calibration data represented by static edges in the SR graph, but also aggregates the transformations from the table to the ART system, the ART system's readings and the transformations from the ART locatables to the real objects. This is modeled by additional edges from the table to the objects.

- Applying the Ubitrack formalism to the SHEEP setup allows to make implicit coordinate system assumptions explicit. For example, in the original version the projection table was implicitly assumed to be at the center of the world coordinate system. It was not documented at which parts of the code this assumption was made, thus minor changes such as making the projection table movable were not possible without excellent knowledge of all parts of the application's code. The green edges in figure 3.16 show that using the Ubitrack formalism allows to model a moving projection table in a clear and understandable way. Even without knowledge of the application's code, it is clear that the software must be changed only at two places: first, an inference must be made that aggregates the spatial relationship between the ART system and the table locatable and the statically measured relationship between this locatable and the table itself and then inverts this relationship in order to compute the function q^{inv}_{TA} associated with the edge e_{TA}. Second,

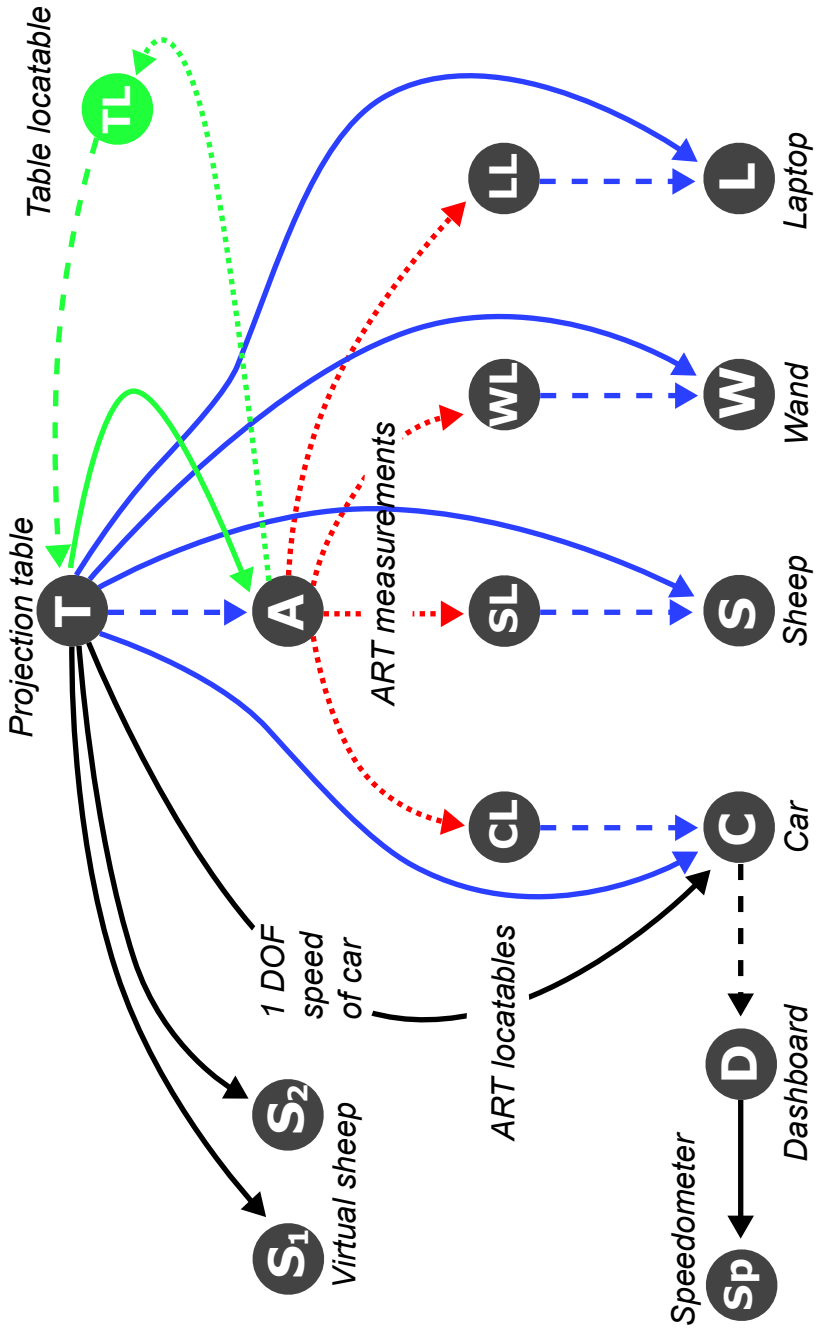


Figure 3.16: The spatial relationship graph of the SHEEP demo. If not noted otherwise, all spatial relationships have a state space of 6 DOF pose. Dashed lines indicate static data, solid lines indicate spatial relationships that change permanently. Blue lines indicate relationships estimated by the *ObjectCalibration* software. Red dotted lines indicate estimates done by the ART software. The green lines to and from the table locatable TL show the simple modifications necessary for a tracked projection table.

the *ObjectCalibration* software has to be modified such that it takes the spatial relationship between the table and the ART system not as a static value, but rather as the constantly updated input from the software just described. Besides that, not a single piece of the demo must be modified.

3.4 Limitations

The major design goals of the formal Ubitrack model were implementability, an extensibility of application domains and state space, resusability of existing fusion components and a careful treatment of time. They were all reached to a certain extent, however, the formal model just introduced clearly has some limitations.

The measurement state spaces that can be expressed within the model must always be of geometric nature. Many ubicomp applications, especially in the commercially promising domain of location based services, use methods such as spatial indexing to derive information such as "User J is in room 501". Common ubicomp location sensing methods such as RFID tags only yield such information. In addition, containment relationships such as "Room 501 is on floor 4 in building IN" are of great value to many applications. It is not clear how these spatial relationships can be expressed within the Ubitrack formal model.

From the point of view of implementability, the very general evaluation function is a great challenge. How should applications provide such a function? By means of an abstract language that then gets compiled by the runtime environment? By handing over binary code? Or by choosing one of a fixed number of potentially parameterized predefined functions? All solutions are not completely satisfying. Additionally, the general pathwise definition of the evaluation function is of little use in practice, due to the number of paths between two nodes being exponential in the number of nodes of a graph.

The extensibility of the formal model to new application domains is ensured by the attribute describing the quality of measurements and the evaluation function. If we assume that some setup already exists and should now be reused by a new application, we will usually be tied to the "old" attribute set. Unless some standards will emerge over time, the flexible choice of attribute sets will be a major problem when existing sensor setups should be reused or combined with other existing setups.

CHAPTER 4

A Distributed Implementation Concept

Overview

This chapter discusses a conceptual implementation that allows to apply the Ubitrack formalism to real world problems.

Reconsidering the requirement of dynamic integration of mobile setups at runtime, I argue that only a peer to peer based implementation fulfills our needs. For such a solution, two problems have to be solved. First, the strategy of how to distribute the available spatial information has to specified. Second, a distributed algorithm for searching optimal paths in the SR graph has to be designed. Viable solutions to both problems are given.

In real-world applications it can be assumed that both the spatial relationship graph's topology and the attributes of individual edges in the graph change much less frequently than the spatial relations between objects. If this holds true, a two phase approach is possible: first, an optimal path between two nodes has to be discovered. From then on, a component making use of the transitivity of spatial relations continuously aggregates the sensor data in real time along the path just found. Although highly flexible in configuration, the aggregation component does not add any runtime overhead compared to existing static solutions.

The chapter is concluded by a discussion of the problems arising out of the two phase approach. The assumptions made may lead to suboptimal solutions of the Ubitrack problem at runtime, however, the expressive power of the formalism is not reduced by the proposed concept. Solutions to how path search results can be cached and reused, and when already found shortest paths have to be reevaluated, will be presented.

4.1 Distributed Peer-to-Peer Architecture

Following the vision of ubiquitous computing, an implementation should allow the integration of a vast amount of sensors, both mobile and stationary. Given the scenario of an intelligent environment equipped with sensing and display technology and enabling its users to naturally interact with the infrastructure, a centralized approach is reasonable. A central server could hold the complete representation of the SR graph, fulfilling all search requests and eventually setting up an inference component with the correct configuration. In addition, this approach would facilitate recomputation, as the information triggering it would be stored at the same place at which the optimal path search would be executed.

The situation gets different if users have their own equipment such as cameras on their cell phones or wearable sensors. The resulting capability of ad-hoc connections of new sensors and sensor networks is a key requirement for the Ubitrack problem.

I propose a fully distributed hybrid peer-to-peer approach. Although it is much more difficult to design and implement, I think the following advantages make a clear point for going in this direction:

Allowing ad-hoc connections between mobile setups: If we treat every network node uniformly, we get ad-hoc connectivity between mobile setups for free. Thus, the computing infrastructure gets truly ubiquitous, even allowing two users meeting on the green field to use the sensors of their counterparts without any additional infrastructure.

No single point of failure: In a centralized approach with a server in the stationary environment, a failure of the infrastructure renders all mobile and stationary sensing technology useless. This may lead to severe implications in case of an emergency, when sensors might be used to compute optimal escape routes. In the proposed distributed approach, parts of the overall sensor network can still be used, especially self-powered mobile setups.

Mobile setup is self-contained: If a user has a mobile setup with more than a single sensor, he can get sensor fusion results without making use of a central infrastructure. This is of particular importance in emergency situations, where some location information from local sensors is still better than none. In addition, a user might use sensing technology as well outdoors, without any supporting infrastructure.

Enabling privacy: With a peer-to-peer implementation the user has full control over the data his sensors gather, he can use it without transmitting data to anyone else. This enables privacy and security mechanisms.

A motivating factor for choosing a peer-to-peer distribution architecture was the

experience gained from developing the DWARF system, that follows a similar distribution philosophy.

4.2 Distribution Architecture

In a distributed sensor network for location tracking, each sensor is attached to one of multiple host computers. Special software components running on these hosts take the sensor data and interpret it such that some spatial relationship between two objects can be derived. Speaking in terms of the Ubitrack graph, these software components correspond to edges in the spatial relationship graph. Figure 4.1 shows an example involving four hosts.

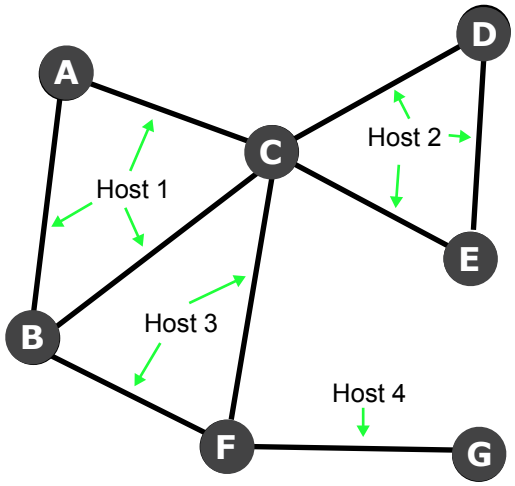


Figure 4.1: A spatial relationship graph describing a distributed sensor network. Every edge corresponds to some software component delivering interpreted data from some sensor. Every software component is executed on some host in the network.

As such, there is no direct representation of a SR graph node in a distributed sensor network. Knowledge about edges can only be derived by observing which software components are running. To minimize replication of data, I propose a hybrid peer-to-peer approach with a centralized component running on every host computer, called the *Ubitrack Middleware Agent (UMA)*. The UMA gathers knowledge about all components providing spatial relationships that are running on the same host. Using the locally available information on edges in the SR graph, a UMA can construct a subgraph of the SR graph that consists only of edges that are associated with local software components and nodes that are adjacent to these edges.

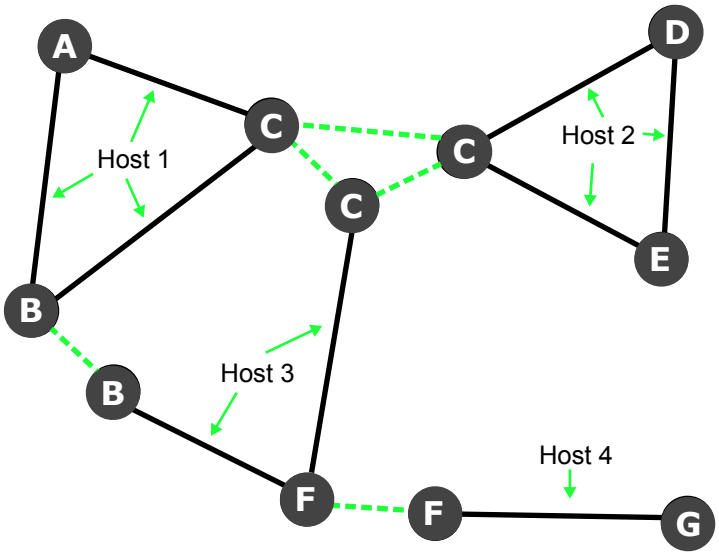


Figure 4.2: Distributed storage of SR graph. Several nodes are replicated, thus zero-weight edges have to be added to allow an optimal path search. Note that edges are not replicated.

If this subgraph construction is done on all hosts in the network, several SR graph nodes get duplicated, as can be seen in figure 4.2. To allow a path search nevertheless, we conceptually have to add zero-weight edges between all nodes representing the same node in the SR graph describing the overall setup. As we will see in the next section, the path search can then be implemented by asynchronous message passing along these zero-weight edges connecting the network nodes.

Prerequisites. To make the proposed distribution architecture work, two requirements must be fulfilled.

Globally unique node IDs: For a UMA to construct a local SR subgraph, all node names must be locally unique. Otherwise, it would not be possible to decide whether two edges have a common adjacent node. To make the inter-UMA communication work, it must be possible for a UMA to identify other UMAs having information about some locally available node. This can only be guaranteed if nodes in the SR graph have globally unique IDs. For fixed setups, this requirement is easy to fulfill. In highly dynamic environments with potentially unknown users, some bootstrapping mechanisms have to be provided that are beyond the scope of this thesis.

Locating arbitrary nodes: It must be possible to find out which host has information about a SR graph node with a given ID. This essentially leads to the general

service discovery problem that is attacked by several solutions, such as Universal Plug And Play (UPNP)[1], Zeroconf[2] or the Service Location Protocol[3]. For the work described in this thesis, it is assumed that service discovery is available.

Both requirements are easy to fulfill in any experimental setup. However, if the distributed architecture is to be deployed in a large environment with many mobile users, some additional effort has to go into adopting naming schemes and service location techniques.

4.3 Distributed Path Search

According to the distribution architecture, we have a partition of the SR graph consisting of several subgraphs that are distributed among networked hosts. Every host only has connections to other hosts that have common nodes. Some application running in this network issues a query for an optimal spatial relationship between two nodes X and Y according to an evaluation function e. According to the generic transitivity-based algorithm 1, the implementation has to detect an optimal path between the given nodes.

4.3.1 Prerequisites

As discussed in section 3.2.8, first finding *all* paths between two nodes and then selecting the optimum is not feasible in practice, as the number of paths is exponential in the number of a graph's nodes. In contrast, shortest path algorithms have a polynomial running time and are a well-researched topic. The prerequisite we impose upon the formal model is that *only edgewise evaluation functions are allowed*.

The formal Ubitrack model does not specify any actual choice of measurement state spaces and attributes describing measurement and inference properties. To make any path search possible, we must restrict the possible state spaces and attributes to those where algorithms for three tasks can be given:

Transitivity of spatial relationships: Given the spatial relationship between object A and B and the relationship between B and C, compute the relationship between A and C. For the standard 6 DOF pose state space used in common AR applications this is fairly simple, if we use a 4×4 homogeneous matrix representation of spatial state, this can be done by a single matrix multiplication.

[1] `http://www.upnp.org/`
[2] `http://www.zeroconf.org/`
[3] `http://www.ietf.org/rfc/rfc2608.txt`

Transitivity of attribute sets: Given the attribute sets of edges AB and BC, compute the attribute set of the edge AC representing the spatial relationship derived in the previous step. This can be a much more complex task, some examples for propagation rules are discussed in section 3.2.6. In the case of attributes describing the error of measurements or inferences this leads to solving equations for error propagation, based on a suitable error model. It may be possible that the application requesting the relationship AC is not interested at all in a subset of the attributes. Then the members of this subset need not be computed explicitly, it suffices if they are somehow taken into account by the evaluation function.

Symmetry: Given a directed edge AB, compute the inverse edge BA. Again, for the spatial relationship with a 6 DOF pose state space this is fairly simple, in case of a homogeneous matrix representation it leads to matrix inversion. For the attribute set, we can make essentially the same observations as with the transitivity of the attribute set – inverting edges is a complex task. If we want to reflect the numerical errors introduced by matrix inversion in the attribute subset describing the error of the inference, things get even more complex.

The transitivity requirements are essential to make any inference, the symmetry requirement is necessary to allow a shortest path algorithm to explore all paths in any connected component. It can be used to construct a directed graph that only has pairs of edges in opposing directions.

4.3.2 Distributed Algorithm

In section 3.2.8 the two major classes of shortest path algorithms were sketched. For the label-setting algorithms (such as Dijkstra's), a single node has to be chosen at every round. After the round, this node has its final label. Obviously, some synchronization is needed in a distributed implementation of a label-setting algorithm. In contrast, the label-correcting class of algorithms (such as the Bellman-Ford algorithm) is easier to implement in a distributed setting. Following a proposal by Beyer [17], we chose an asynchronous variant of the Bellman-Ford shortest path algorithm introduced by Lynch [74].

Asynchronous Bellman-Ford Algorithm. The algorithm assumes that every node of the graph corresponds to a node in the network. Network nodes communicate by exchanging *messages* asynchronously. The limited capacities of real-world computer systems are modeled by two kinds of FIFO queues. Every communication channel with restricted capacity has an associated FIFO queue. A communication channel needs at maximum some time d to deliver a single message, if it has to deliver k

messages, it consequently needs time $k \cdot d$. Analogously, a network node with limited processing power has an associated FIFO queue to sequentially process incoming messages. Again, if a message needs at maximum time l to be processed locally, the processing of k messages needs time $k \cdot l$. The FIFO queues come in handy for modeling message pileups due to limited processing power or communication channel bandwidth: if a message arrives and we already have a pileup of $k-1$ messages, it takes a worst-case time kd to process the new message. The number of nodes is n and the number of edges is m.

For communication, every network node i has several *input channels* taking a current estimate w of the shortest path from the source node to a neighboring node j:

$$\text{rcv}(w)_{j,i}, \quad w \in \mathbb{R}, \quad j \in \text{neighbors with edges to } i$$

In addition, every network node also has several *output channels* sending the current label w (i.e. the currently best estimate of the node's distance to the source node) to a neighboring node k:

$$\text{send}(w)_{i,k}, \quad w \in \mathbb{R}, \quad k \in \text{neighbors with edges from } i$$

Every node keeps its current distance d to the source node i_0. At the start of the execution all nodes have distance $d = +\infty$, except the source node which has distance $d = 0$. Every node also keeps a pointer to a *parent node* p, i.e. the node preceding itself on the currently shortest path from the source node to it, initialized with *null*. Finally, every node i has to keep several FIFO queues q_j of tentative shortest distances, for every communication channel with neighboring nodes j that have edges to i. At initialization, only the queues of i_0 are initialized with a single value, 0.

At runtime, every network node in parallel receives and sends messages. The main loop is depicted in algorithm 4.

Modifications to the Algorithm. The basic algorithm just described must be enhanced by several modifications to make it useful in practice.

Ensure termination: A process in the network never knows when it can stop waiting for new messages. Technically speaking, the current algorithm is not yet correct, because it never can produce the output (essentially making a final decision on the values of p and d). However, this can be ensured by convergecast acknowledgments. Whenever a node receives a message, it has to signal the sender of this message when all actions resulting from this message have terminated. If the message does not lead to a distance update, the acknowledgment can be sent immediately. Otherwise, things are more complicated: after the update, messages will be sent to all neighboring nodes. Only after these messages have been acknowledged themselves can the originating node

Algorithm 4: Distributed asynchronous label-correcting shortest path algorithm

```
parallel
    begin
        while Some shortest distance queue q_j is not empty do
            w ← first element of q_j ;
            send(w)_{i,j} ;
    end
    ;
    begin
        while Some message recv(w)_{j,i} is received from node j do
            if w + weight(e_{j,i}) < d then
                d ← w + weight(e_{j,i}) ;
                p ← j ;
                foreach k ∈ neighbors − j do
                    add d to q_k, i.e. trigger notifications of neighboring nodes ;
    end
end
```

be signaled. Some bookkeeping operations are necessary. Yet, the complexity of the algorithm is only increased by a constant factor. The algorithm terminates when the source node i_0 obtained acknowledgments for all messages it initially sent out. It can then send termination messages to all other nodes via network flooding.

Graph partitioning: The basic asynchronous algorithm is suitable for pure peer-to-peer based networks. Our distribution architecture is not pure, but hybrid peer-to-peer, with a centralized UMA running on every network node and controlling a subgraph of the SR graph. This fact can be used for speeding up the overall computation. At its first participation, every network node solves the all-pairs-shortest-path problem for the subgraph it keeps with the given evaluation function. Both the evaluation function's values applied on all edges and the shortest paths and distances between any two nodes get stored. Whenever a message from some other UMA is received for a specific SR graph node, all locally available nodes can be updated very efficiently using these precomputed values. Care has to be taken at the UMA carrying the start node. After solving the APSP problem, all nodes with links to other UMAs have to be initialized with the shortest local distance to the start node. Their message queues have to be initialized with this value as well in order to start the algorithm.

Propagation of shortest path between two nodes: Although the algorithm computes the distance between the source node and all other nodes, in a sensor fusion query issued by an application only the shortest path between the source and a single target node is of interest. The search starts at a UMA having information about the start node, and it would be favorable if the same UMA finally could obtain the resulting shortest path. This can be done in a simple fashion: When the target node gets the termination signal, it sends a message to its parent node p. Subsequently, each grandparent node sends a message containing the path from itself to the target node until the source node is reached. Note that an edge in the SR graph corresponds to some software component, as such, the IDs of these components need to be transmitted. The source node UMA can then set up an inference component with the given information.

4.3.3 Complexity Analysis

The complexity analysis for the chosen algorithm can be divided in two parts: time and communication complexity. First Lynch's analysis [74] of the basic algorithm will be summarized and then the influence of our modifications will be detailed.

Basic Algorithm: Exponential Worst Case Running Time. Unfortunately, the asynchronous algorithm has both a worst case time and message complexity that is exponential in the number of nodes. This is due to the fact that we can not assume an upper bound on the time a single message takes, instead, pileups of messages may occur. For every FIFO queue q in the algorithm, we can only guarantee that the time to deliver the oldest message is at maximum d, thus, message delivery of a new message takes up to time kd if the queue already contains $k-1$ messages.

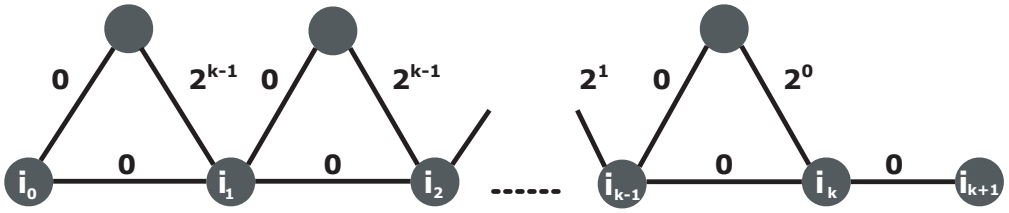


Figure 4.3: Graph leading to pileups in message queues of asynchronous Bellman Ford algorithm.

Lower bound: For every even node number $n \geq 4$, a graph G can be constructed according to figure 4.3 (in the picture, $k = \frac{n-2}{2}$) that makes the asynchronous Bellman Ford algorithm send at least $\Omega(c^n)$ messages and takes at least time $\Omega(c^n d)$ to stabilize in the worst case, for some constant $c > 1$.

In this graph, all edge weights are 0, only the right-facing sloped edges have weights that are decreasing powers of 2. We now assume that the communication channels associated with the "flat" zero-weighted edges are very slow, and the "sloped" edges have very fast communication channels. After initiating the algorithm, the node i_k gets the first message along the upper paths, updating its distance label to $2^k - 1$. Next, it gets a message from node i_{k-1} along the zero-weight flat edge, leading to a new reduced distance label $2^k - 2$. Now node i_{k-1} gets a message from its neighbor i_{k-2} along the lower path, leading to a distance label reduced from $2^k - 2$ to $2^k - 4$. Node i_{k-1} sends this reduced label to node i_k on both paths. The message via the upper path arrives first, reducing i_k's distance label to $2^k - 3$, afterwards the message via the lower path arrives, again reducing i_k's distance label by 1, resulting in the new value $2^k - 4$.

If we further assume that all channels operate very quickly compared to the channel from node i_k to the target node i_{k+1}, then messages pile up, leading to a queue of 2^k messages in this channel, which is $\Omega(2^{\frac{n}{n}})$. Thus, a lower bound on the message complexity is $\Omega(c^n)$.

If all these messages take the maximum time d to be delivered, the overall running time is $\Omega(c^n d)$, which is a lower bound on time complexity.

Upper bound: The number of messages sent to a node along an edge is smaller or equal to the number of distinct simple paths between the source node and the given node. Thus, an upper bound on the message complexity is $\mathcal{O}(n^n m)$. Consequently, an upper bound on the time complexity is $\mathcal{O}(n^{n+1}(l + d))$.

Note that it is possible to optimize the FIFO queue behavior by restricting its length to 1, i.e. we use the fact that messages put into the queue have a strictly decreasing distance label associated with them. However, the worst-case running time is still exponential, as the time for "sending a message" is now reduced to the time for putting a message into the queue, which is still greater than zero.

Influence of Modifications. Termination of the algorithm is guaranteed by acknowledgment messages for all messages received by other nodes. This modification adds a constant factor 2 to the algorithm's message complexity c_m, and $c_m d$ to the time complexity. Sending the final termination signal from the source node via network flooding takes m messages taking maximum time md.

Transmitting the shortest path information back to the source node takes a maximum of m messages, and maximum time md.

Finally, partitioning the graph and solving the APSP problem takes worst-case time $\mathcal{O}(n^3)$ in case the Floyd-Warshall algorithm (see section 3.2.8) is used. No messages are exchanged for this purpose. However, in practice we can assume every

network node to have a limited maximum edge number, leading to constant effort for the preprocessing step.

Complexity of Service Discovery and Connection Phase. The algorithm discussed depends on the availability of suitable service discovery techniques. The following steps of the proposed modified algorithm require service discovery:

Locating the source node: Whenever an application emits a query, it must send this query to the source node of the path search. Thus, the application (or the middleware) must locate this node.

UMA interconnection: Due to the hybrid peer-to-peer distribution architecture, all network nodes with UMAs running on them must have connections to other UMAs with adjacent edges. In the worst case, $\mathcal{O}(m)$ such connections have to be set up.

Setting up inference components: The path search returns a list of edges of an optimal path. If this list contains just an abstract representation of edges, the inference component must locate all corresponding components. However, a more efficient implementation returning pointers to these components in the edge list makes this location step superfluous.

Complexity of Evaluation Function Propagation. The evaluation function of the formal Ubitrack model is not restricted, as such, arbitrary functions can be associated with any query. These functions have to be transmitted to each network node, and each node has to apply the function to each edge. Thus, an additional complexity term of $\mathcal{O}(m)$ is necessary for the evaluation function, if it is assumed to be of constant size.

4.3.4 Discussion

The exponential worst case running time of the proposed algorithm is a major problem when trying to deploy the proposed algorithm in large-scale ubicomp environments. We chose it nevertheless, due to its simplicity, the simple mapping of a natural distribution architecture on the algorithm and ease of implementation. In average case, the algorithm performs much better, in fact is has served as ARPANET's routing algorithm between 1968 and 1980 ([74], page 506). We can thus expect it to scale to a certain extent.

Restricting Queries to Subgraphs. Still, improvement is possible. An obvious source of guaranteeing reasonable running times is to restrict search queries to subgraphs of the overall SR graph. The motivation is as follows: an application is

usually interested in a spatial relationship with an error that is small compared to the extent of the spatial relationship. Thus, for queries relating two continents, an acceptable absolute error of the result might be some kilometers, for queries relating two objects within a single room, the absolute error should be in the centimeter range. In consequence, it is possible to prune the SR graph to "relevant" trackers, in our example, the long-distance query does not have to take centimeter-accuracy location sensors into account, whilst the local query may only use sensors mounted in the room the two objects are in. In addition, the overall number of SR graph edges to be combined by any query result is limited. Errors within the underlying measurements accumulate quickly such that we can expect the overall error to grow beyond a reasonable extent for long paths in the SR graph.

Setting up clear definitions, implementation concepts and prototypes for such a *locale* based partitioning of the SR graph is major future work.

Synchronizing the Algorithm. As Lynch points out, synchronizing the proposed algorithm speeds up its worst case massively. If we have a global synchronization, rounds of the algorithm can be assured. A maximum of $\mathcal{O}(n)$ rounds is necessary, as after round i, all shortest paths up to length i are guaranteed to have been found. At every round, a message can be sent along every edge, leading to a message complexity of $\mathcal{O}(nm)$ and a time complexity of $\mathcal{O}(n(d+l))$.

However, the effort to synchronize the algorithm is not neglectable. In small networks (as they only might occur if we restrict queries to subgraphs), the overhead for synchronization might be higher than the saved average case running time.

Using a Label-Setting Algorithm. Träffs [121] gives distributed asynchronous implementations of label-setting shortest path algorithms, based on Dijkstra's algorithm with a heap implementation. He reports polynomial worst case running time, however, the analysis is based on the unrealistic assumption that communication takes unit time. If we impose this assumption on our label-correcting algorithm, its complexity gets reduced to $\mathcal{O}(n)$.

4.4 Dynamic Spatial Relationships

The Ubitrack formal model has the power to express all spatial relationships of a certain setup within a well-defined time interval of arbitrary length. Although a very long interval might be favorable for some applications, most often it will only lead to an overly complex spatial relationship graph that contains much unused information.

For example, consider a setup within a hallway that is used by many people. A vision-based tracking system observes people walking through it. As such, every new user observed is represented by a new node in the SR graph and an edge

between the user and the tracker node, associated with some spatial relationship function. This function is only defined within the time interval the user walked through the hallway. Following the Ubitrack formalism strictly, after several days of use, the SR graph will consist of a large number of user nodes, which might lead to storage and access problems.

In our example, it seems advisable to delete a user's node some time after the user left the hallway. Formally, this corresponds to allowing only a subset of all potential nodes and edges in the SR graph. We define this as follows:

- A *time-restricted SR graph on the interval* $[t_1; t_2]$ contains only edges with associated spatial relationship functions that are defined within a subset of $[t_1; t_2]$.

- It contains only nodes with at least a single adjacent edge.

Most often, the time interval will have the form of a sliding window, i.e. it will be of fixed length and store some past (for history information needed by some filtering schemes) and some future (for prediction) information. As such, the time interval will be centered around the current time t: $I_{\text{slide}} = [t - t^-; t + t^+]$

Using the concept of time-restricted SR graphs introduces new challenges for any implementation: it must handle a graph with ever changing topology. In addition, the interval length must be chosen carefully to prevent a repeated insertion and deletion of nodes and associated edges of location sensors with a low update rate.

4.5 A Two-Step Approach to Inferring Knowledge

Some of the requirements derived in section 1.4.2 are opposing each other. On the one hand, an implementation must not add significant overhead to the overall latency in a distributed tracking setup, and on the other hand, it must allow a highly dynamic reconfiguration at run time of a potentially very large number of sensors.

All algorithms described in this chapter have been designed to implement the generic transitivity based algorithm developed in section 3.2.7. In short, the SR graph is taken as is. To compute an inference of the spatial relationship between two objects represented by nodes X and Y, a single optimal path between these nodes is selected according to criteria defined by an evaluation function e. Spatial data is inferred along this path. The generic algorithm does not make use of knowledge from any other path or subpath between nodes X and Y, as additional world knowledge would be necessary for this step.

4.5.1 Naive Implementation

A naive approach would be based on some representation of the current SR graph. Whenever an application needs a spatial relationship between two nodes, it issues a query to the naive Ubitrack implementation. The implementation then finds all paths between the two given nodes (or does a shortest path search), applies the evaluation function to each path, selects the path with the smallest evaluation value, propagates the attributes and spatial relationships along this path and returns the propagated values to the application. If this operation is executed atomically, the application gets an optimal estimate.

In practice, such an implementation will only work for a very limited number of nodes or applications at a very low update rate and with a very high latency, due to the computational overhead for the path search and attribute propagation. In fact, it would be unusable.

4.5.2 Dividing Connection Setup and Communication

In existing static tracking setups, only the spatial relationships and some attributes (e.g. tracking accuracy) get computed out of several measurements each time new data becomes available. This obviously can be done sufficiently fast to fulfill the requirements of typical AR applications.

In consequence, a two-phase process in order to reach a sufficient real-time performance is proposed.

Phase 1: Connection Setup. Upon receiving an application's request for a spatial relationship between two objects X and Y being optimal according to an evaluation function e, the implementation finds an optimal path according to e between the nodes representing X and Y. It then configures an *inference component* that takes the values of the measurement and/or inference functions q_{input} associated with the edges in the optimal path as input. The inference component yields the aggregated spatial relationships and attributes along the optimal path, i.e. it corresponds to the function $q_{\text{inference}}$ associated with a new inferred edge e_{XY}.

Phase 2: Runtime Communication. Whenever an application requests an update to the spatial relationship, the inference component gathers the data from all components representing the functions q_{input}. It then aggregates the spatial relationships and the attribute set. The result of this aggregation is delivered to the application.

Using this approach, the overhead resulting from using a dynamic formal model just occurs at selected instances in time. After the setup phase, the implementation is as fast as a handcrafted equivalent. In addition, it fits well into the formal model

by providing an inference component that can be associated with a new edge in the SR graph.

4.5.3 Implicit Assumptions

The two-phase implementation concept is based on some restricting assumptions. The spatial relationships are assumed to change much faster than both the attributes describing their properties and the topology of the SR graph.

Stable SR Graph Topology. The optimal path search is only performed once, and consequently happens on a snapshot of the SR graph. If edges get added or deleted at a later point in time, there might be a new optimal path between some nodes, making a recomputation necessary. This might happen if some sensors get switched on or off, if mobile equipment is brought into the tracking environment or taken from it and if some sensors fail. If a recomputation has to be done almost as often as a computation of aggregated spatial relationships, the overhead resulting from setting up an inference component leads to an overall efficiency that is worse than that of the naive approach.

In practice, the assumption of a stable SR graph topology holds true most of the time. Current sensor setups consist of a relatively small number of sensors, and mobile users with their own sensor setups have to be integrated rather seldom. In addition, the search space for optimal paths can usually be restricted to a relatively small spatial entity (e.g. a single office). It makes no sense to take into account an inference that involves several long-distance trackers if the desired spatial relationship can also be computed out of some locally available knowledge. Restricting the search space to a small spatial entity eventually leads to a small SR graph. The smaller the graph, the lower the possibility that the topology changes.

Stable Attributes. As with the graph topology, the attributes also influence the results of an optimal path search. If attributes taken into account by the evaluation function change, a recomputation of the path is necessary to ensure its optimality. Indeed, many attributes such as pose error covariances for vision-based trackers change with every single measurement. Strictly speaking, a recomputation of the optimal path is required whenever a new measurement occurs, degrading the efficiency to an unacceptable level. In consequence, the two-phase approach can not guarantee optimal results at all times.

Again, in practice this is less of a problem. In common setups, some location sensors of highly varying accuracy are available, e.g. a wide-area tracker such as the Bat system [2] and a narrow range device such as a magnetic tracker. Although the attributes describing the accuracy of both devices might change with every measurement, they still differ by an order of magnitude, supporting our opportunistic

choice of the optimal path based on a snapshot of the attributes.

If two sensors with attributes in the same order of magnitude of error are installed stationarily in the same place, it seems advisable to handcraft a filtering component that uses additional knowledge of the sensor characteristics in order to obtain results superior to each individual sensor. This filtering component would then have attributes that are almost always superior to each individual sensor's attributes, leading to an automatic choice of it with the optimal path search algorithm.

Recomputation Implemented. There exist several possibilities how the recomputation of optimal paths can be triggered. The most simple approach is to recompute paths at fixed time intervals. This may lead to wasted computational resources, but is very easy to implement. A more sophisticated solution would trigger a recomputation whenever the topology of the SR graph changes, making some notification mechanism necessary. For every inference component, the optimal path needs to be stored, and whenever a change in topology occurs, this stored path needs to be compared with the result of a recomputation. If both differ, the inference component needs to be reconfigured.

For the reasons described above, a recomputation upon every change of attribute is not feasible. However, it would be possible to define both a subset of relevant attributes (such as a binary "lost track" flag) and some thresholds of changes that might trigger recomputations.

4.6 Results of Path Search Implementation

The exponential worst-case running time of the proposed algorithm for detecting optimal inference paths makes a practical implementation necessary to prove the feasibility of the approach. In this section, the results of an implementation of the search algorithm that was applied to several random distributed graphs are discussed.

4.6.1 Related Results.

Träffs compares his label-setting algorithms on a 15 processor transputer system linked in a 4×4 quadratic mesh [121]. He uses a general purpose routing system that makes communication expensive compared to computation, a situation similar to our scenario of mobile and stationary ubicomp computing devices linked via wireless networks. Two graph types are evaluated: random unstructured graphs with a predefined node degree and "grid graphs" organized in layers with a maximum number of nodes each, with the source node being a member of the first layer.

To summarize his findings, notable speedups up to 4 can be achieved for random unstructured graph, regardless of their size and average node degree. In the case of

grid graphs, no speedup can be found, instead the overall running time is increased by 50%. However, problems that could not be solved on a single computer for insufficient memory reasons can now be computed using the parallel implementation.

4.6.2 A Simulation Environment for the Distributed Path Search Algorithm

The major goal of the implementation described in this section is to separate the effects of service discovery and connection setup from the actual running time of the path search algorithm. For this reason, a simulation environment for asynchronous distributed algorithms is described. It is used to apply the proposed path search algorithm to random graphs and evaluate its performance

Key Concepts. The key idea of the simulation environment is to provide a sequential, single-threaded execution for an asynchronous message-passing algorithm. A ring buffer modeling *global time steps* is used for this purpose. With every entry of this buffer, a set of an arbitrary number of messages is associated.

Network hosts are modeled as objects that have a *local time*. If a network host receives a message from the simulation environment, it compares the global time stamp with its local time. If the local time is smaller or equal, the host accepts the message and processes it. After processing the message, the host sets its local time to the sum of the received global time and adds the *local message processing time* to it, modeling a consumption of processor time. If the local time is larger than an incoming message's global time, the host rejects the message and tells the simulation environment its current local time, i.e. the next time it will be idle and can accept messages.

The key loop of the simulation environment is described in algorithm 5.

As the simulation environment has full control over all messages, at the end of execution the global time needed for processing and the number of sent messages is obtained.

To model restricted communication facilities and message passing time of arbitrary length, *communication channels* are used to filter a host's incoming message stream. Similar to the network hosts, these channels have a local time and a per-message processing time that is advanced every time a message is forwarded to the corresponding host. If a message's global time stamp is smaller than a channel's local time, the message is not accepted and must be rescheduled by the simulation environment. Communication channels are selected by the simulation environment based on the sender/receiver pair of a message.

Graph Generation and Host Initialization. On startup, the simulation generates a random input graph and associates each of its edges with random weight and a

Algorithm 5: Main loop of simulation environment for asynchronous message passing algorithms.

```
while receivedHaltMessage == false do
    while ringBuffer(globalTime%BUFFERSIZE) contains a message do
        msg ←
        ringBuffer(globalTime%BUFFERSIZE).removeNextMessage ;
        if msg is halt message then receivedHaltMessage ← true
        msg.timeStamp ← globalTime;
        host ← msg.receiver;
        hostTime ← host.receive(msg);
        if hostTime > globalTime then
            /* Reschedule Message                                    */
            ringBuffer(hostTime%BUFFERSIZE).add(msg);
    globalTime ← globalTime + 1 ;
```

random host. Then all hosts are initialized such that they have a representation of their local subgraphs according to the distribution strategy described in section 4.2. The hosts also know adjacent hosts for every local graph vertex, i.e. which other hosts have duplicates of it. Finally, every host precomputes a all-pairs-shortest-path matrix for its local subgraph to speed up future computations.

The simulation environment then selects two random source and target vertices of the global graph and uses Dijkstra's algorithm to compute a shortest path between them. This path is used as ground truth later to verify the distributed solution.

Setup Phase. The simulation chooses a host keeping a copy of the search start vertex and sends a *start* message to this host. Upon receiving this message, the host sets the start vertex' distance label to zero, notifies all other hosts having copies of the start vertex about the new distance label and updates its internal vertices' distance labels.

Runtime. At runtime, the simulation environment executes the main loop described in algorithm 5. This leads to sending messages to hosts that then execute the path search algorithm described above, including the necessary bookkeeping of acknowledgment messages to ensure termination.

As soon as the start message gets acknowledged, the simulation environment main loop returns.

Termination. Now a *get result* message is sent to the start network host, and the simulation main loop is started again. The start host broadcasts this message via flooding to all other network hosts.

As soon as a network host controlling a copy of the target node receives the termination message, it starts assembling the shortest path from source to target in reverse order. The algorithm works as follows:

Procedure `AssembleEndOfPath(`*Vertex target*`)`

Data: *target*: Target vertex
Result: *intermediateSource*: Some vertex along shortest path that is also represented at another host; *Path*: Path

```
begin
    intermediateSource ← null ;
    d_min ← ∞ ;
    forall graph vertices i that are also existent at other host do
        if APSP(i, target) < d_min then
            intermediateSource ← i ;
            d_min ← APSP(i, target) ;
    Path ← Dijkstra(intermediateSource, target) ;
end
```

A message containing *Path* is sent to the host that is stored as predecessor with vertex *intermediateSource*.

Intermediate network hosts, i.e. nodes that neither control the source nor the target vertex, operate as follows when they receive a partial shortest path message from another network node at vertex i_t:

Procedure `AssembleIntermediatePath(`*Vertex target, Path receivedPath*`)`

Data: *target*: first known part of received shortest path
Result: *intermediateSource*: Some vertex along shortest path before *target* that is also represented at another host; *Path*: the shortest path from *intermediateSource* to the original target node

```
begin
    intermediateSource ← predecessor(target) ;
    Path ← Dijkstra(intermediateSource, target) + receivedPath ;
end
```

Again, a message containing *Path* is sent to the host that is stored as predecessor with vertex *intermediateSource*.

If the network host controlling the source vertex s receives a message with the shortest path between some local vertex i_t and the target node t, it solves the SSSP problem for s and appends the path from s to i_t to the received path to get a shortest path between s and t. The simulation environment subsequently leaves the main loop and the shortest path result can be output along with the number of time steps and messages that were necessary for its computation.

Implementation Details. The simulation environment was implemented in Java. The data structures of the graph library JGraphT[4] were used to store the global and local graphs.

Random numbers were generated using Java's `Math.random()` method. Random graphs were generated according to the $G(n, p)$ model. A random graph $G(n, p)$ has n vertices, and every edge between two arbitrary vertices i and j has the probability p to exist. To model the edge inversion resulting from the symmetry requirement stated in section 4.3.1, only pairs of edges (i, j) and (j, i) were allowed, albeit with differing weight.

4.6.3 Practical Results.

The simulation was run on random graphs, and results were averaged over several shortest path queries. The number of network hosts, number of graph vertices and the probability of edges were varied. Every network host was modeled to have a single input communication channel, thus emulating the single network interfaces of typical mobile computers. The distributed path search algorithm's local processing time can almost be neglected compared to the message passing time in real-world setups. Thus, the local processing time was set to 1 and the message processing time was set to 100 via the communication channel's parameters. The result graphs shown below give the overall processing time in multiples of the time it takes to process a single message.

Figure 4.4 shows the dependency on the number of hosts for a fixed-size graph with 50 vertices and on average 5 adjacent edge pairs per vertex. A speedup of 2 can be observed if the number of hosts increases from 5 to 25.

Figure 4.5 shows how the number of graph vertices influences the number of messages and the computation time. In the simulation, both grow almost linearly with the number of graph vertices.

Figure 4.6 show how the graph density influences the time and message complexity. Up to a edge probability of 0.5, the complexity grows massively, but the growth curve flattens significantly for denser graphs.

Discussion. The simulation results show that the proposed distributed algorithm is viable for small-scale setups. The time complexity profits from an increasing number of network hosts to a certain degree, and the approach even scales to dense graphs which are very uncommon in typical Ubitrack setups.

Yet, the high number of messages every network host has to process require a highly efficient implementation. In the case of figure 4.6, the shortest path computation consumes as much time as transmitting 4000 messages. If we assume MacWilliams' measurements of the DWARF CORBA-based AR middleware [77]

[4] `http://jgrapht.sourceforge.net/`

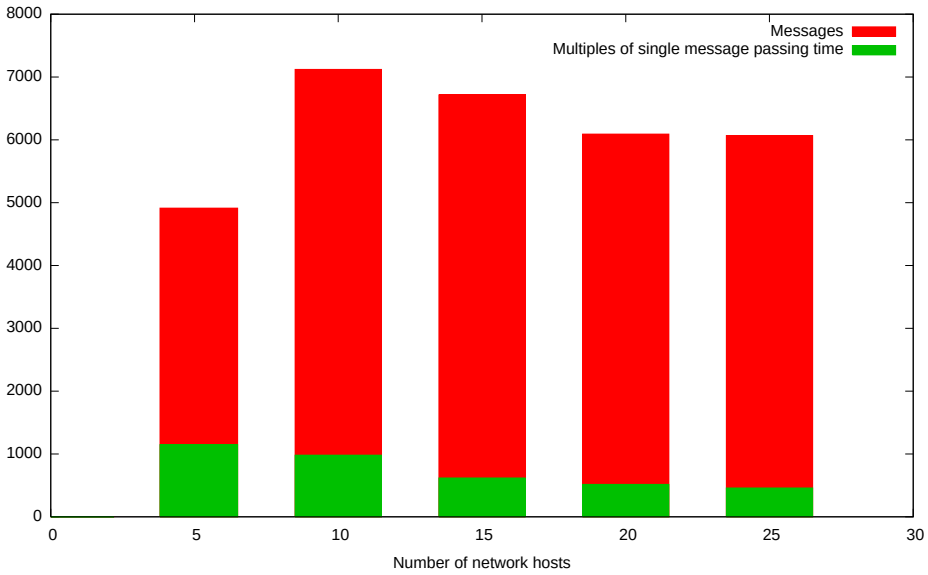


Figure 4.4: Simulation result with number of network hosts as a parameter. The underlying graph had 50 vertices and on average 5 pairs of edges adjacent to every vertex.

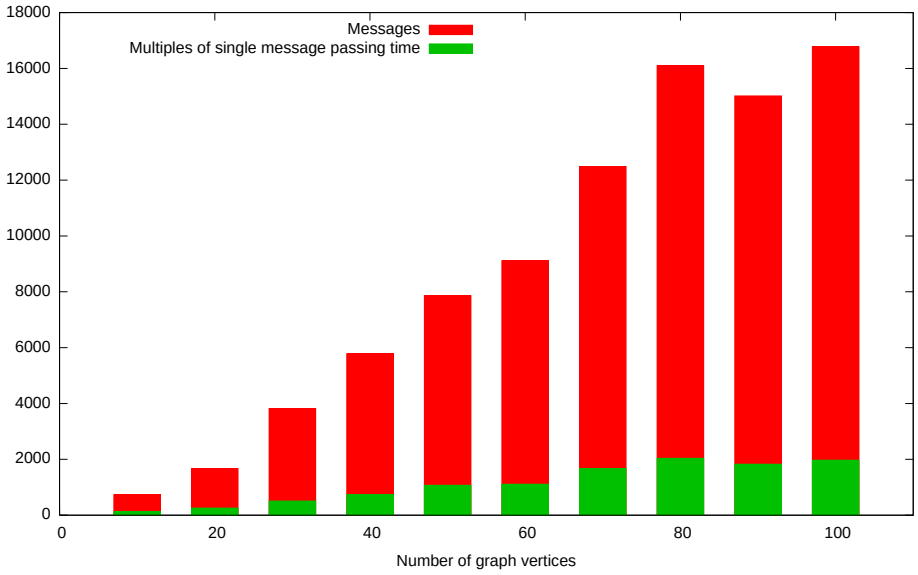


Figure 4.5: Simulation result with number of graph vertices as a parameter. The random graph had on average 5 pairs of edges adjacent to every vertex and was distributed on 10 hosts.

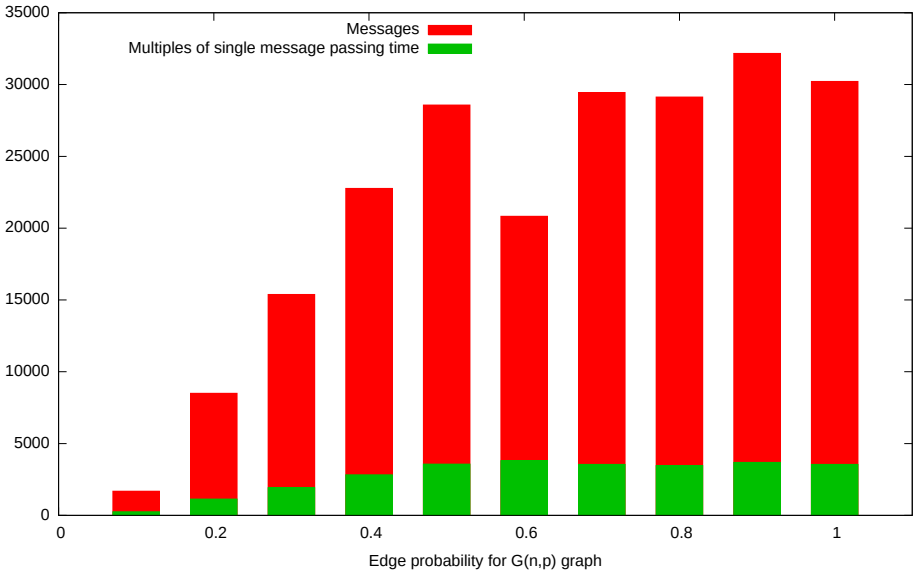


Figure 4.6: Simulation result with edge probability as a parameter. The random graph has 50 vertices and is distributed on 10 hosts.

which takes 7ms to transmit a message wirelessly, the path computation would take 28s, which is beyond usability.

4.7 Inferring Knowledge along Optimal Paths

After the search for an optimal path is finished, the proposed implementation concept sets up a runtime *inference component* that continually aggregates spatial data along the optimal path.

This section discusses several issues that have to be considered for every actual implementation of this runtime infrastructure.

Transforming the SR Graph into a Data Flow Graph. Conceptually, the process of setting up an inference component is the transformation of a subgraph of the spatial relationship graph into a data flow graph for aggregation of spatial data. Consider the simple AR setup depicted in figure 3.6. It consists of three objects, T, L and O and two measurement functions $q^m_{TL}(t)$ and $q^m_{LO}(t)$. Function $q^m_{TL}(t)$ corresponds to a static relationship, function $q^m_{LO}(t)$ models the output of a location sensor. Three inference functions are constructed, namely $q^e_{TL}(t)$ and $q^e_{LO}(t)$ for extending the time domain of the measurement function, and $q^i_{TO}(t)$, aggregating the spatial relationships of the q^e functions.

Figure 4.7 shows the resulting data flow graph. At runtime, data from both the component holding the static calibration $q^m_{TL}(t)$ and the component encapsulating

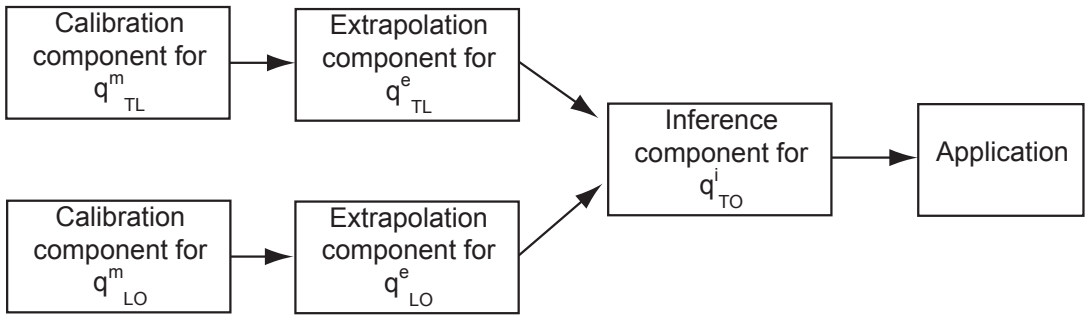


Figure 4.7: Data flow graph resulting from simple AR setup depicted in figure 3.6.

the location sensor modeled by $q^m_{LO}(t)$ is delivered to components that perform the time domain extension. Data from these components flows to the component computing the aggregation $q^i_{TO}(t)$, that delivers the result to the application.

The modular inference concept resulting from this approach allows further optimizations: if multiple applications need spatial data that gets inferred along partially overlapping paths, parts of inference components may get reused. However, the general optimization problem arising from these questions is beyond the scope of this thesis.

Mapping Data Flows on Actual Implementations. Every data flow architecture (e.g. OpenTracker [99]) could be used for the execution of the data flow graph. However, the recomputation issues discussed in section 4.5.3 pose additional requirements. It must be possible to reconfigure an inference component at runtime, triggered by new inference possibilities. There must also be some feedback mechanism for the inference component to trigger a recomputation of an optimal path, in case some necessary input component fails.

The next chapter discusses a possible implementation based on the DWARF AR framework.

4.8 Discussion

This chapter presented a highly distributed asynchronous algorithm to implement the generic algorithm for finding optimal inferences. It works in a peer-to-peer fashion, allowing the ad-hoc connection of mobile setups. The worst-case runtime of the algorithm is exponential in the number of network hosts. Yet, a simulation gave promising results for small-scale networks. The runtime efficiency of the proposed concept can be enhanced massively using a two-step approach: first, a shortest path between two SR graph vertices is computed, second, a run time data flow between

distributed components is set up. The implicit assumptions of stable SR graph topology and attributes were discussed.

There are several observations to be made about the proposed concepts:

Changes in SR graph attributes: Strictly speaking, most attributes of edges in the SR graph are changing with every measurement. Especially for AR applications, measurement error is the most relevant attribute [76]. Yet, the error is always dependent on a particular measurement and thus changes with every new measurement. Nevertheless, for the reasons discussed in section 4.5.3, we can assume the attributes to be stable in most real-world tracking setups.

Scalability: The price for the proposed algorithm's flexibility regarding ad-hoc networks is its worst-case exponential running time. Thus, the size of the graph to be searched has to be restricted to allow the proposed concept to scale. This can be realized by pruning the graph according to contextual knowledge. For example, if the optimal spatial relationship between two objects in the same room is to be evaluated, it usually suffices to scan only the parts of the SR graph in this room.

Dependency on service location: The proposed concept depends on an efficient service location protocol, which is beyond the scope of this thesis. However, the service location problem is not yet solved in a satisfying and massively scalable way. Again, in real-world setups this is less of a problem, as the small size of such setups allows using a variety of existing service location techniques.

Suboptimal results: Due to the dynamic nature of the underlying SR graph to be searched, the distributed search algorithm is not guaranteed to give optimal results. For example, it might return a shortest path containing an edge that corresponds to a tracking device that was switched off during the path search. In practice this problem can be remedied by recomputing shortest paths at fixed time intervals.

CHAPTER 5

An Implementation based on DWARF

Overview

The preceding chapters specified a formalism that allows to model arbitrary networks of location sensors and an abstract distributed implementation concept that only marginally restricts the expressive power of the formalism. This chapter describes a working implementation based on the DWARF AR framework that serves as a proof of concept of both the formalism and the implementation concept.

After an explanation of the key concepts of DWARF which are necessary for implementing the distributed Ubitrack concept described in the last chapter, this chapter discusses how the elements of the formalism can be applied to theDWARF concepts. In principle, the Ubitrack spatial relationship graph gets mapped on a corresponding DWARF service connection graph. Most important, edges in the SR graph are mapped to nodes in the DWARF service graph, as they are represented as software components delivering the spatial relation between two objects. Properties of these objects are also modeled as DWARF services.

The actual implementation consists of a Ubitrack specific extension of the DWARF service manager [77], the DWARF Ubitrack Middleware Agent (UMA) that employs the algorithms described in section 4.3.2 to infer application requested spatial knowledge from the available sensor data. Similar to the DWARF service manager, there is one UMA per network node, thus allowing an implementation of the distribution strategy described in section 4.2.

5.1 DWARF Concepts

The *Distributed Wearable Augmented Reality Framework* (DWARF) is a research platform under constant development at Technische Universität München since 2001. This section briefly lists the requirements influencing crucial design decisions, explains the key concepts of the DWARF approach to building AR systems and details some recently added features that support dynamic environments like the scenarios of ubiquitous tracking.

5.1.1 Dwarf Requirements

The key idea of DWARF is to use a set of interoperating *distributed components* to model AR applications. This can be motivated by looking at the requirements arising from various views on creating AR systems [12, 98].

Project Manager's View. A project manager responsible for an AR system has to keep cost and time constraints. If a system can be split into components, these can be seen as black boxes that only are accessed via well-defined interfaces. Component development and testing can then be distributed in space and time, leading to a lowered interdependency of various teams responsible for different subsystems.

An additional advantage arises if some generic general-purpose components are provided. Then, a new AR system can be authored by simple reconfiguration of preexisting components. Another advantage is the possibility of "rapid prototyping" [78] for quick testing of new ideas.

To summarize, the requirements for a component architecture from the project manager's view are mainly a simple configurability of components and well-defined interfaces for communication between components.

User's View. The user's view on an AR system in intelligent environments focusses on ease of use. With a component-based approach, an AR system's functionality can be split into several functional units like communicating with a cell phone, rendering of virtual objects using a head mounted display or tracking a user's pose with a location sensor.

In current AR systems, installing a new tracking device is a task that can only be performed by experts. With the component-based approach, the necessary software (including interactive calibration routines) could be bundled with tracking hardware, thus facilitating the setup to a mere plugging of the new device into the existing component network.

In summary, an AR system's user benefits if components are used to abstract hard- and software to functional units.

Application Developer's View. An application developer is used to structure complex systems into several layers, thus encapsulating basic functionality and making it accessible to high level parts of the system.

Reicher [98] proposes to describe the architecture of AR applications in four layers, namely the architectural style, the domain-specific architecture, the product-line architecture and the application architecture.

Thus, the DWARF framework has to be modular enough to allow a layering of components as necessary for an application developer to reuse major parts of previous efforts.

Component Developer's View. A developer of a new component wants to make use of as many existing components as possible. Thus, it should be possible to specify in an abstract fashion the data a component needs to work properly and the data it is able to deliver to other components.

In addition, a component developer puts the requirement that a middleware allows the automated connection of components, including network transparency and a choice of communication protocols that are of use in AR applications, such as event-based communication or shared memory blocks for exchanging high-bandwidth data such as video streams locally.

5.1.2 Dwarf Key Features

DWARF can be split in several pieces, namely the *middleware* implemented with the DWARF *service manager* providing transparent access to individual components, called DWARF *services* and a set of *design concepts* on how to create AR applications in a well-defined fashion.

The DWARF *service manager* sets up a hybrid peer-to-peer [110] network. An instance of the service manager is running on every network node, and DWARF services connect to it via CORBA[1]. The service manager controls its local services and maintains descriptions of them. Each service manager cooperates with the others in the network to set up connections between services.

A DWARF *service* is the basic building block of a running system. It either encapsulates a hardware device like a location sensor, performs some reusable functionality like controlling a taskflow or handles some application-specific task. Each service is running within a separate operating system process or thread. Its functionality is described in terms of *abilities*, the functionality it requires from other services is described in terms of *needs*. Needs and abilities are matched using *connectors*.

An *ability* is the abstract description of a service's functionality, e.g. spatial data for trackers. A service can have multiple abilities, e.g. the tracker may track

[1] `http://www.corba.org`

multiple objects simultaneously. Abilities are typed, a tracker delivers `PoseData` for location information.

A *need* describes the functionality a service requires from its counterparts to be able to work. Again, a service may have multiple needs. A vision-based tracker needs a stream of video data and descriptions of markers to be able to find these markers in the video image. Needs are typed, too, and a need can only be satisfied by an ability of the same type.

A *connector* is a description of the communication protocol, e.g. shared memory for video data, CORBA object references or CORBA notification events for event-based communication of location data. It is the responsibility of the service managers to set up connectors for matching services. After the connectors have been set up, the services can communicate directly without any involvement of the service managers. This two-step connection setup fulfills the usually contradictory flexibility and efficiency requirements for systems combining ideas from ubicomp and AR.

Attributes enhance the description of an ability. The tracker may therefore specify which object it tracks, using e.g. `Target=JoesHead`. Needs can be refined using *predicates*, e.g. `(&(User=Joe)(Room=Lab))`. When matching needs with abilities, the service managers ensure that the ability's attributes satisfy the need's predicate. Attributes may be specified for the entire service, in this case all abilities of that service have these attributes.

```
<service name="OpticalTracker">
    <need name="video" type="VideoStream">
        <connector protocol="SharedMemory"/>
    </need>
    <need name="marker" type="MarkerData"
        predicate="(&(Room=ARLab)(User=Joe))">
        <connector protocol="ObjectReference"/>
    </need>
    <ability name="poseData" type="PoseData">
        <attribute name="Room" value="ARLab">
        <attribute name="User" value="Joe">
        <connector protocol="NotificationPush"/>
    </ability>
</service>
```

Figure 5.1: Sample XML description of an optical tracker service having two needs of type *VideoStream* and *MarkerData* and an ability of type *PoseData*. Only *MarkerData* abilities of other services with specific attributes (`Room=ARLab` and `User=Joe`) are requested, and the *PoseData* ability of this service is attributed accordingly.

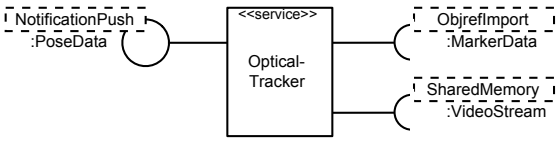


Figure 5.2: Sample UML description of an optical tracker service. Needs are depicted as semicircles, abilities as circles.

Each DWARF service installed on a network node may either describe itself at startup to the service manager or use an XML file format to give the service manager the possibility to start and stop the service on demand. Our example of an optical tracker is shown in figure 5.1. A UML-based notation is shown in figure 5.2.

5.1.3 Supporting Dynamic Environments

A major design goal of DWARF is to support dynamic environments as they occur within common ubicomp scenarios. The DWARF key features just described are well suited for static setups, and we have implemented several demonstration applications [12, 78] that were configured statically. To support dynamic environments, the topology of the service connection graph needs to be changed at runtime. Several concepts within DWARF support this requirement.

Starting and Stopping Services. The DWARF service managers have knowledge about each service's current communication partners. To support power management scenarios on wearable computing platforms, a service manager can notify a service if it is accessed for the first time or if it is not used any more. This behavior can be triggered by setting the flag `startOnDemand` in the XML service description.

In addition, service managers constantly ping services under their control and other service managers to detect whether they are still available. This mechanism allows the detection of changes in the topology of the service connection graph.

Sessions. Every need description can be enhanced by describing the desired multiplicity, using the tags `minInstances` and `maxInstances`. Specifying a minimum number of instances (default value is 0) causes the service to be started only after this minimum number of matching abilities becomes available. For example, a service transforming a sensor's base coordinate system makes only sense if the sensor is actually available, thus `minInstances` would be set to 1. The maximum number of instances can be specified to restrict the number of connections either for logical reasons (for example, a viewer component needs exactly one service specifying the virtual camera's viewpoint) or to limit the load on a system. Abilities can not be restricted in their multiplicity using the service description.

DWARF employs a *session* concept to handle multiple connections to a need or ability. If a service object implements the CORBA `SvcSession` interface, consisting of the methods `newSession(...)` and `endSession(...)`, these methods get called before another service gets connected. Based on the descriptions of the connecting service given as arguments, the service object can decide which local object should handle the new session and return this (potentially specifically instantiated) object to the service manager.

Using sessions, a service can dynamically keep track of where it sends to or receives from data.

Changing Service Descriptions at Runtime. Every service can query its service manager for the service description. It is also possible to *change* this description at runtime, by modifying attributes of the service and existing needs and abilities or by adding or removing needs and abilities.

For example, a video grabbing service wrapping some frame grabbing hardware and putting digital images in a shared memory buffer might set the attributes describing the properties of the digital image (e.g. the color model) according to some parameters it obtains from the frame grabber hardware. Additionally, it might detect automatically whenever someone plugs in a new camera and then set up a new `VideoData` ability.

Changing a service's description can lead to the necessity for breaking existing or setting up new connections. This is handled automatically by the service managers.

Template Services. The concept of template services allows to define generic components that can be configured at runtime.

Two classes of services can be distinguished. *Singleton services* exist only once for a given service description, all needs, abilities and attributes are fixed. For example, the video grabbing service described above is a singleton service.

In contrast, *template services* exist in multiple instances. For example, a filter service that smooths pose data might exist simultaneously for multiple pose data streams. It is also possible that a service's ability is defined as a template. For example, a generic fiducial marker tracking service that takes a video image out of a shared memory segment and detects a certain number of fiducial in it, can then provide one *template ability* for pose data relating the camera's and the fiducial's pose for each fiducial. In the XML service description, the flag `isTemplate` indicates that a service or an ability can be instantiated multiple times.

The service manager is responsible for creating new service or ability instances whenever it detects possible communication partners for a service's needs. It is possible to bind the service's or ability's attributes to attributes of the partner service's ability that is satisfying the service's need. For example, the optical tracker service described in the last section has a need for `MarkerData` with the contex-

tual predicates `Room` and `User`. The service uses the information obtained via the `MarkerData` need to provide a `PoseData` ability that again has the attributes `Room` and `User`, set to the same values as in the corresponding need. In the example, `Room` was bound to `ARLab` and `User` was bound to `Joe`.

```
<service name="OpticalTracker">
    <need name="video" type="VideoStream">
        <connector protocol="SharedMemory"/>
    </need>
    <need name="marker" type="MarkerData"
        predicate="(&(Room=*)(User=*))">
        <connector protocol="ObjectReference"/>
    </need>
    <ability name="poseData" type="PoseData" isTemplate="true">
        <attribute name="Room" value="$(markerData.Room)">
        <attribute name="User" value="$(markerData.User)">
        <connector protocol="NotificationPush"/>
    </ability>
</service>
```

Figure 5.3: Sample XML description of an optical tracker service with a template ability of type *PoseData*. If the need for *MarkerData* gets satisfied by another service with attributes *Room* and *User*, the optical tracker offers the *PoseData* ability with the same attributes.

To describe the optical tracker in a generic fashion, the ability `PoseData` must be templated. Figure 5.3 shows the XML notation used for template services.

The new services and/or abilities resulting from template instantiations remain as descriptions inside the service manager and are only started by launching a new process or thread within an existing process if another service has a need for them. After startup, the new service or ability connects to its communication partners with the specified communication protocols and works independent from the service managers.

Chains of Services. The template service mechanism can also be used to set up chains of services. For example, an application might have a need for 6 DOF pose data of a user's head. The available services consist of a frame grabber service providing video streams, a simple generic marker tracker processing video streams and extracting 6 DOF pose information, a marker configuration service storing information how to track the user's head optically, an inertial tracking service providing high-update 3 DOF orientation information, and a generic filter service

combining 6 DOF pose with 3 DOF orientational information, yielding 6 DOF pose information with higher update rate and accuracy in the rotational part.

As can be seen in figure 5.4, after the connection of all services, the generic services are fully configured.

It should be noted that the general problem of finding suitable chains of services for a given setup is not solvable in polynomial time, as such, it seems only reasonable to use this concept for specific tasks.

5.2 Mapping the Ubitrack Formalism on DWARF

The DWARF platform has been used to create a proof-of-concept implementation of the distributed concept described in chapter 4. For this purpose, crucial parts of both the Ubitrack formalism and the distributed implementation concept have to be mapped on DWARF concepts. This section describes this mapping.

5.2.1 General Concepts

The Ubitrack formalism proposes to use a graph for modeling knowledge about spatial relationships in an intelligent environment. A DWARF runtime system consists of a graph of interconnected services that are distributed as well. Consequently, some mapping from the Ubitrack SR graph onto the DWARF service graph has to be defined. To make the implementation feasible in real-world applications, it must support the concept of time-restricted SR graphs defined in section 4.4.

Layered Architecture. The Ubitrack implementation in DWARF can be regarded as an abstraction layer from all low-level location services.

Figure 5.5 shows the concept of the layered Ubitrack architecture. Services encapsulating location sensors are accessed from the Ubitrack layer using a *sensor API*. The Ubitrack layer refines the sensor data streams and delivers the inferred data to arbitrary applications. Note that an application might also be another DWARF service that needs spatial relationships to work properly, e.g. a natural feature tracker that needs to initialize itself. Applications access the Ubitrack layer via a *query API*.

SR Graph Edges. Spatial relationships are expressed as functions associated with edges in the SR graph. The implementation of such a function in software results in some component. As a consequence, SR graph edges get mapped on DWARF services, i.e. nodes in the DWARF service connection graph. Several classes of spatial relationship services can be distinguished:

Location sensor: A service of the class *location sensor* encapsulates some hardware device and induces its measurements interpreted as spatial relationships. In

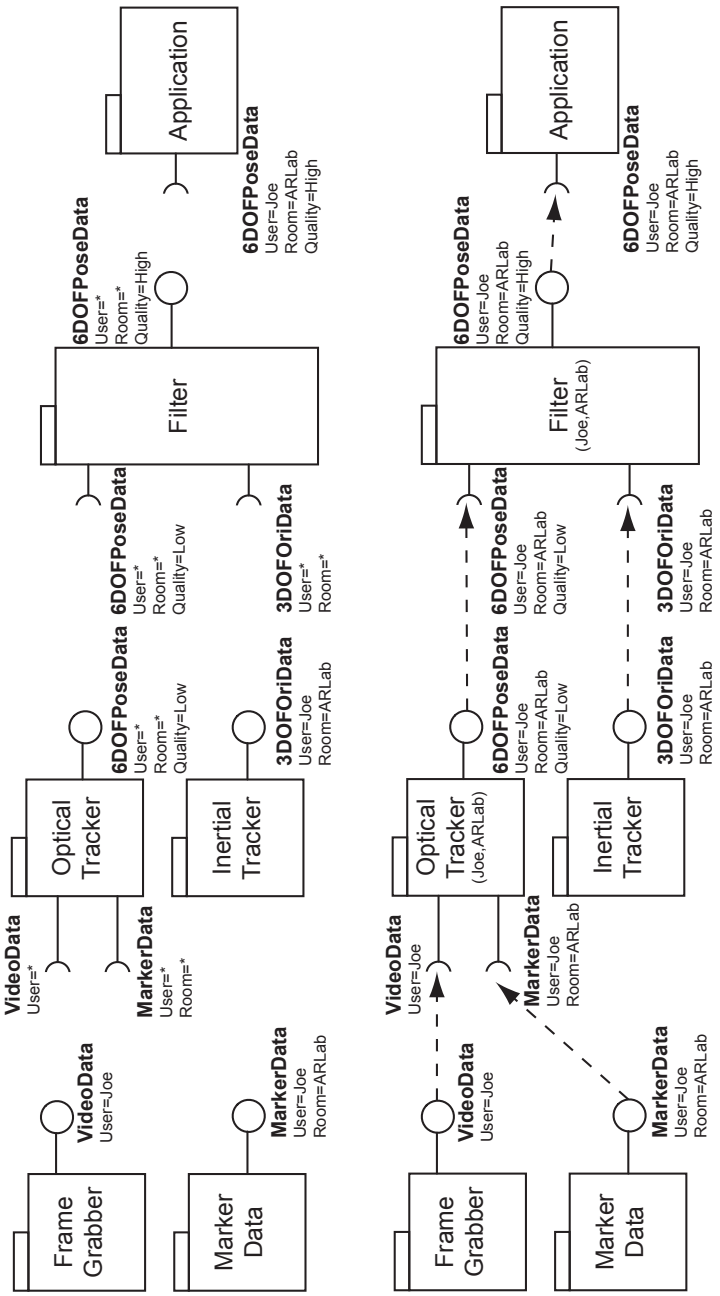


Figure 5.4: A chain of services. At the top, the unconnected services are shown, with both the marker tracker and the filter service being underdefined. At the bottom, the same services are shown after connection, now the marker tracker and the filter services are fully specified. If there was another marker data ability, new instances of the marker tracker and the filter service could be instantiated. Note that the attribute `Quality=Low/High` is just for illustration. In real applications, some parameters according to a specific error model would be used.

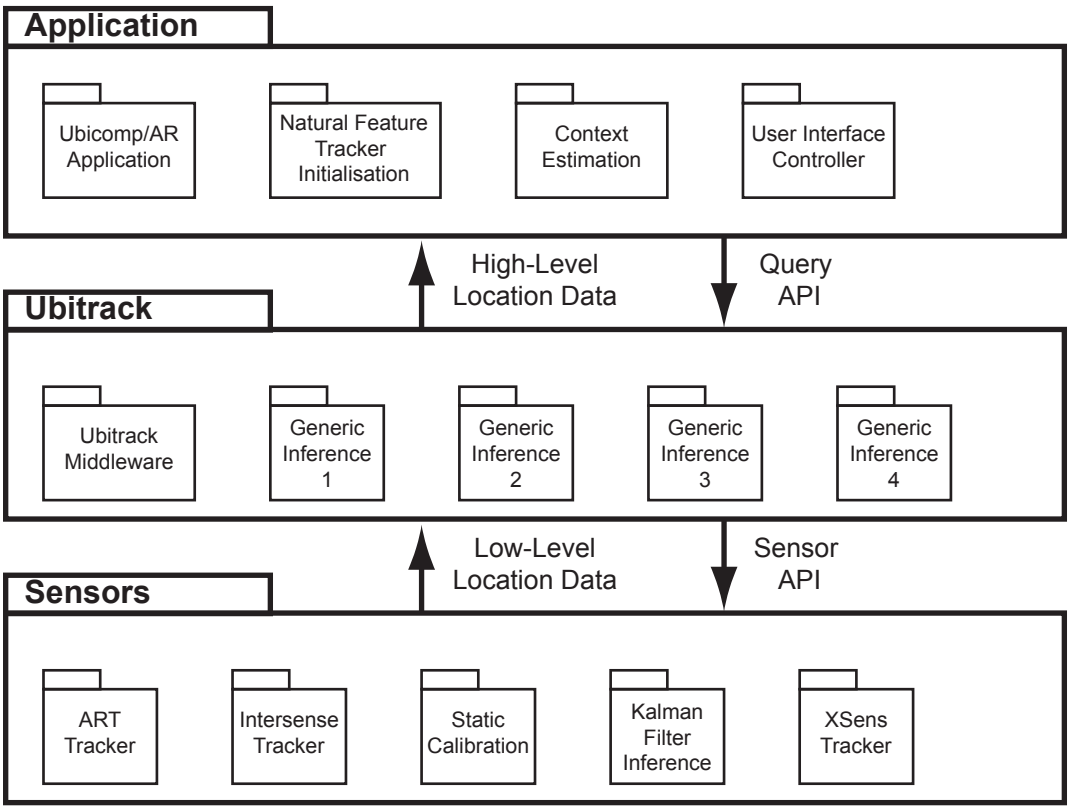


Figure 5.5: Layered Ubitrack architecture. The Ubitrack layer is an abstraction from the low-level location sensors and provides a powerful API to arbitrary applications.

DWARF, several location sensor services exist, such as services providing data from Intersense trackers or an ARToolkit-based optical tracker.

Static calibration: In real-world AR setups, several spatial relationships, e.g. between a locatable and an object it is attached to, must be determined in some off-line calibration procedure [123]. These relationships are estimated once and then do not change over time. DWARF *static calibration services* model such calibration results, usually being fed from files or a database [120].

Special inference: In complex sensor fusion systems, algorithms such as a Kalman filter process the data of several location sensors. These algorithms get encapsulated into *special inference* services.

Generic inference: Some inferences on spatial relationships can be generalized to a certain extent. For example, a function to change the reference coordinate system of an object's 6 DOF pose is simple to implement and parameterize.

Generic inference services provide such functions. In contrast to the other low-level service classes, generic inferences are part of the Ubitrack layer.

The connections between DWARF services, i.e. the edges in the service connection graph, depend on the needs of the individual services. In general, both location sensor and static calibration services only have outgoing edges, i.e. deliver data to other services. Inference services have both in- and outgoing edges, as they process data received from other services and send the results to services of a higher layer. Formally, whenever a function associated with an edge e_1 in the SR graph and implemented as service s_1 serves as parameter of another function associated with e_2 and implemented as s_2, the corresponding DWARF service connection graph has an edge from the node representing service s_1 to the node of service s_2.

SR Graph Nodes. The nodes of the Ubitrack SR graph represent objects in the virtual or real world. For the purpose of spatial relationships, they only have to carry a unique ID, that can also be extracted from the set of the graph's edges. Consequently, there is no necessity to model SR graph nodes within the DWARF service concept. However, an object's properties might be of interest to some AR or ubicomp application. For example, a "where's my friend" people finder application might use pictures associated with SR graph nodes that represent people. For this purpose, it is possible to model SR graph nodes as DWARF *object description services*.

Distribution Architecture. DWARF has the same hybrid peer-to-peer architecture as the Ubitrack distributed implementation concept. Consequently, full access to a DWARF service representing some SR graph edge is available only to the local middleware. The DWARF service manager's service location facilities are used for setting up the necessary connections between the network nodes and locating start and target nodes of graph search requests. The messages of the distributed path search algorithm are exchanged using event channels.

Dynamic Behavior. The dynamic topology of a time-restricted SR graph gets mapped on the DWARF service connection graph using the features described in section 5.1.3. Whenever a new location sensor, static calibration or inference service becomes available, the information about it is available at the local network node and, using the DWARF service discovery techniques, at the next distributed path search. If a service becomes unavailable, e.g. after unplugging a tracking device or if a user leaves an intelligent building, the service manager network is notified of this event and adapts accordingly.

5.2.2 APIs of the Ubitrack Layer

To make the layered approach work, two APIs need to be defined. The *sensor API* is for unified access to all kinds of location sensors, and the *query API* is for abstract access to spatial relationships from applications.

DWARF offers two main facilities for API definitions. First, using the concepts of needs and abilities along with attributes and predicates, services can either specify requests by dynamically creating a need for some spatial relationship specified by a set of predicates or offer some facility by creating an ability that is described by a set of attributes. Second, communication mechanisms such as event channels or remote procedure calls can be used for a more direct communication.

Sensor API. It is the task of the sensor API to specify a common data format for expressing all spatial relationships as modeled in the Ubitrack SR graph. Obviously, this data format is tightly coupled with the state space of the spatial relationships. In the current DWARF implementation, only a 6 DOF state space has been considered.

The API consists of two parts: first, so-called `PoseData` structures sent via event channels or remote method calls carry the parts of a SR graph function that changes rapidly. The elements of this data structure are shown in table 5.1. Second, DWARF service *attributes* of the ability representing a sensor store the parts of the function which have a low update rate. The ability's type is `PoseData`, therefore expressing the sensor's state space. Some service attributes must be given, others are optional. The required attributes are `UTSource` and `UTTarget`, optional attributes might be selected out of the examples discussed in section 3.2.6.

Query API. The query API's task is to provide unified abstract access of applications of any kind to all available spatial relationships. In the DWARF implementation, an application has to have a *need* according to the type definition of a certain state space. In the current implementation, only the 6 DOF absolute position and orientation `PoseData` type is provided. The need must be enriched by *predicate* specifying the kind of spatial relationship an application is interested in. The predicate must be composed using the following attributes:

UTSource: A string containing the ID of the source coordinate system. This is the source node ID of the corresponding edge in the SR graph.

UTTarget: A string containing the target object's ID.

Other attributes may be used, the following have a specified meaning:

UTUpdateRate: Specifies the desired update rate of the spatial relationship in Hz. Defaults to the underlying sensor's update rate.

Element	Description
source	A string containing the ID of the source coordinate system. This is the source node ID of the corresponding edge in the SR graph.
target	A string containing the target object's ID.
position	An array of three double values containing the 3D position of the object in a Cartesian coordinate system.
orientation	An array of four double values containing the orientation of the object represented as unit quaternion.
covariance	A 6×6 matrix of double values representing the Gaussian error in the position and orientation, according to the error model of section 2.7.3.
timestamp	A data structure containing the absolute timestamp of the measurement, consisting of the standard UNIX time format of *seconds* since January 1st, 1970, 00:00 and *milliseconds*. Note that time synchronization between networked computers has to be ensured to render the time stamp information useful in distributed setups.
timeerror	A double value representing the standard deviation of the time stamp.
confidence	A double value in the interval $[0; 1]$ containing the probability that this measurement exists. Used if a tracker is unsure about the identity of a tracked object.

Table 5.1: Elements of the PoseData structure.

UTTimeOffset: Based on the time the query is released to the Ubitrack layer, this value gives the offset to the desired time at which the spatial relation should be evaluated. Positive values indicate future points in time (thus requiring prediction components), negative values past points in time. If `UTUpdateRate` is not zero, the given offset is constantly applied to the initial time plus multiples of the update interval. A pair of seconds and milliseconds. Defaults to zero.

UTEvaluationFunction: This value gives a specification of the evaluation function to be applied. In the long run, this predicate will consist of a highly complex XML-based description of evaluation functions. In the current implementation, this predicate is not evaluated, as only a single evaluation function exists. It is explained in section 3.2.5.

Finally, an application can express the desired type of communication by specifying a set of *connectors*:

Asynchronous push: This communication type is the most common. Events are sent asynchronously from the Ubitrack layer whenever new data is available, according to the desired update rate. The application must specify a `PushConsumer` connector for this type of communication.

Synchronous pull: This type is based on remote method calls. An application calls a method in the Ubitrack layer, and that method returns with the desired spatial relationship as soon as it has been computed. An application must specify a `ObjrefImporter` connector for this communication type. The given remote object reference of the Ubitrack layer implements the method `PoseData getPoseData(in time : Timestamp)`.

Asynchronous pull: This is the most complex communication type. An application uses a remote method call to send a request, this call starts computation of the desired data and returns immediately. If the computation is finished, an event with the desired data is sent to the application. To make this type of communication work, an application must implement both a `ObjrefImporter` and a *PushConsumer* connector. The resulting remote object of the Ubitrack layer implements the method `void wantPoseData(in time : Timestamp)`.

Note that the actual data containing the spatial relationships is sent using the same *PoseData* structure as described for the sensor API. This allows to reuse inferences done by the Ubitrack layer as input to more complex inferences.

Other Measurement State Spaces. Currently, only a 6 DOF absolute pose state space is supported. If sensors or inference components with other state spaces are to be integrated, the general mechanisms of separating data in two classes according to their change frequency and expressing slowly changing data in attributes and frequently changing data within structures should be kept. Still, suitable sensor error models will have to be found.

Helper Classes. To facilitate the implementation of new components encapsulating sensors or inference algorithms, several C++ helper classes have been implemented. Here some high-level descriptions are given, for details see the DWARF CVS repository[2].

EasyPoseData: This class extends the CORBA `PoseData` structure. It provides simple get/set methods for all data fields of *PoseData* and initializes these with reasonable values. In addition, the methods `inverse` and `product` are offered to facilitate the task of inferring spatial relationship along transitive relationships.

[2] `http://cvsbruegge.in.tum.de/dwarf/`

PoseSender: This class implements the asynchronous push interface. Using its method `sendPoseData`, a programmer can send spatial relationships very easily.

PoseService: This class offers a full DWARF service for sending and receiving pose data. By deriving from this class, programmers can implement sensor and inference components without DWARF or Ubitrack-induced implementation overhead.

5.3 Ubitrack Middleware Agent (UMA)

The DWARF *Ubitrack Middleware Agent (UMA)* [17] is the middleware component providing Ubitrack functionality to DWARF applications. In the current implementation, it offers an ability for 6 DOF spatial relationships of type `PoseData`. An application accesses it via the DWARF need-based query API described above. This section describes the design of this component.

Extending the Dwarf Service Manager. The DWARF UMA extends the service manager. For every service manager running in a distributed DWARF system, a UMA is started as well. The set of UMAs connect to form the same hybrid peer-to-peer network as the service managers, analogous to the distributed Ubitrack implementation concept described in the last chapter.

According to this concept, a UMA has to gather all information about local (i.e. running on the same network host) services providing spatial relationships between real or virtual objects. Out of this information, it constructs the local SR subgraph and connects to other UMAs that have information about common SR graph vertices.

At startup, every UMA connects to the local service manager and uses remote method calls to query for all descriptions of abilities that offer spatial relationships. This is done by requesting all locally available abilities of well-defined types. In the current implementation, only 6 DOF position and orientation data of type `PoseData` can be requested.

To ensure the dynamic nature of the time-restricted SR graph model, the UMA must be notified of changes in local services' descriptions. This is handled by the `ServiceChanged` ability of the service manager, which sends events upon any change in the local set of services.

Creating a Local SR Graph. The UMA uses only service descriptions to create the local SR graph. These descriptions contain DWARF attributes that are used to store SR graph attributes. As mentioned in section 5.2.2, only the attributes `UTSource` and `UTTarget` are required, all other attributes of a service are stored

along with the corresponding SR graph edge and can be used by particular evaluation functions. In consequence, it is the responsibility of a DWARF sensor service's programmer to provide meaningful attributes which are suitable for a particular application domain.

The UMA parses all `UTSource` and `UTTarget` attributes and creates a SR graph node list out of them. The nodes then get connected by attributed edges. On every change of a sensor service, the local SR graph gets updated accordingly.

Connecting to Other UMAs. The distributed path search algorithm discussed in the last chapter relies on asynchronous message passing between the UMA implementations. For the DWARF UMAs, CORBA notification events are used.

There are two types of messages to be exchanged, namely *search requests* that update SR graph node distance labels, and *search replies* that acknowledge the request messages. As every UMA can both send and receive requests and replies, there need to be four unidirectional event channels for every connection between two UMAs: two abilities, one for sending search requests (`SearchRequestSender`) and another for sending search replies (`SearchReplySender`); and two needs, one for receiving search requests (`SearchRequestReceiver`) and another for receiving search replies (`SearchReplyReceiver`).

The abilities get enriched with *multivalued* attributes, containing a list of the IDs of all locally available SR graph nodes. Correspondingly, the needs' predicates narrow the matching abilities to those having information about locally available SR graph nodes. The session concept is used to distinguish between multiple remote UMAs.

Locating Application Queries. As mentioned above, an application sends a query to the Ubitrack layer, i.e. the local UMA, by creating a need for `PoseData`, with the need's attributes forming the query's parameters. The local UMA gets aware of this need using the same mechanisms as it uses to update its information about local sensors. At startup, all needs of type `PoseData` are scanned using the `ServiceManager` remote method call interface, at runtime, the `ServiceChanged` event channel is used to notify the UMA of new requests.

To distinguish needs of applications and of the Ubitrack layer itself for `PoseData`, needs have to be enriched using the predicate `(subsystem=application)`.

As every UMA has information about all local applications' needs, regardless of the start/target nodes in the SR graph they request, it has to forward the SR graph search requests to a UMA storing a representation of the source node. This is done using the message channels described above.

Starting and Observing Inference Components. The final task of the UMA after having found an optimal path between two nodes is to set up an *inference component*

that dynamically aggregates the spatial data available along the computed SR graph path. In the current implementation, this is handled by a UMA *ServiceFactory*.

It uses the connection to the local service manager to specify a new service description that has an ability matching the application need that triggered the search request. In addition, for every edge in the computed optimal path, a need for the corresponding service is added. The inference components described in the next section can parse such a service description and are started automatically using the existing dynamic service manager features.

5.4 Inference Components

As explained in section 4.7, the task of the inference component is to handle the runtime data flow of spatial relationship events. This section describes how this task is mapped on a set of DWARF services.

Data Flow Components. Summarizing issues mentioned in this and the last chapter and applying the usual granularity of DWARF services, the task of inferring knowledge about spatial relationships has been split up in several reusable components [96]:

Inference: This is the workhorse component making the actual inferences along transitive paths in the SR graphs. The implementation essentially consists of code parsing the needs of the service description, and multiplying the runtime 6 DOF pose data. Note that according to the timing discussion of section 3.2.3, all measurements that are aggregated have to be obtained simultaneously. Thus, the connection to other services delivering spatial data is established via the *synchronous pull* interface. Accordingly, the inference service provides data using the same interface.

Kalman filter: This component implements a generic Extended Kalman Filter (section 2.7.1) with a polynomial movement model. It usually receives data via the *asynchronous push* protocol, i.e. by an event mechanism. It provides data to the inference service using the *synchronous pull* interface.

Measurement inverter: This service is started whenever an inverted edge is part of the SR graph path to be aggregated. Remember that we imposed the symmetry requirement on all edges in the spatial relationship graph, and that we construct a symmetric SR graph for the optimal path search. If now a path uses an inverted edge, there is no DWARF service corresponding to this edge. Thus it has to be created by instantiating an instance of the *measurement inverter*.

Measurement sampler: The inference service only offers the *synchronous pull* interface. If an application prefers to get notified using an event mechanism, an instance of the *measurement sampler* has to be inserted at the end of the data flow graph.

Example Data Flow. To make the role of the DWARF Ubitrack data flow components clear, let us reconsider the example "extended tracker range" introduced in section 3.3.1: an ART system A tracks a video camera C that delivers its images to ARToolkit, which in turn tracks a fiducial marker M that is in front of the camera. The offset between the locatable L on the camera and the camera's center point is estimated offline, and the application is interested in the spatial relationship of the marker relative to the ART system's coordinate system, which is the same as that of a projection table P that is eventually used to augment the fiducial marker.

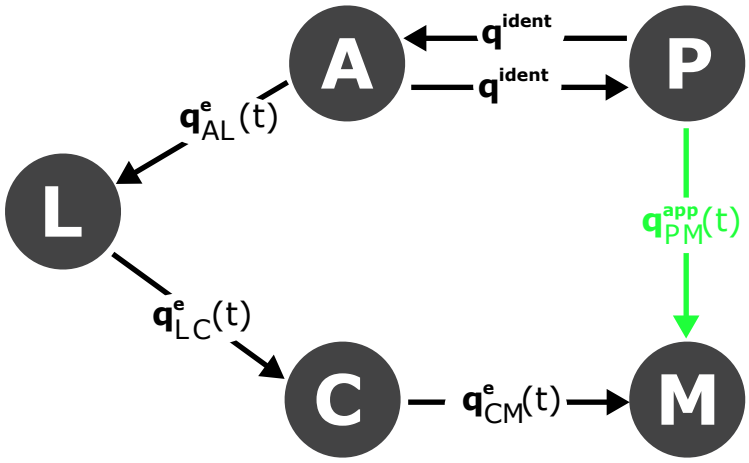


Figure 5.6: SR graph of extended tracker range setup.

Figure 5.6 shows the SR graph of the setup depicted in figure 3.10. Three spatial relationships have to be aggregated dynamically, the dynamic relationships q_{AL} and q_{CM} and the static relationship q_{LC}.

Figure 5.7 shows the resulting DWARF service graph consisting of three Kalman filter services, an inference service and a measurement sampler service. Note that the static relationship q^m_{LC} does in principle not need a filtering component, however, it was added anyway to provide a convenient synchronous pull interface to the inference component.

Extensions. The current implementations' approach gives correct results. Yet, there are some limitations that should be addressed in future revisions. There is no

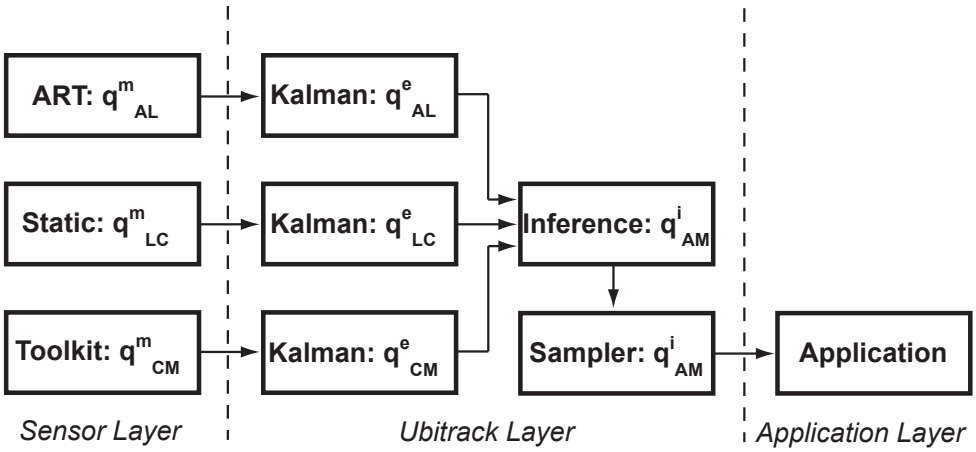


Figure 5.7: DWARF data flow components of extended tracker range example. The diagram shows the SR graph functions corresponding to the individual services and the logical layering of the services. Arrows indicate the data flow.

systematic approach to reusing computations. If, in our example, another application was interested in the relationship between the ART system and the camera, another inference component would have to be started, although the one already running has the desired relationship as intermediate values. The measurement inverter service also introduces some problems. If, for example, a path consists solely of inverted edges, it would be much better to first compute the aggregated spatial relationship in forward direction along the inverted path (i.e. taking directly the available measurements) and only then invert the result. The current implementation uses as many inversion services as there are edges in the path. Finally, the resource allocation for distributed inferences could be optimized. Parts of an inference could be precomputed at some workstation with spare resources, and only the remainder should be done on a low-power wearable computer. Currently, all computations are done on the network host where the application resides, which is a feasible heuristics, as it will minimize network latency in most setups.

5.5 Results

A first prototypical approach towards mapping the Ubitrack concepts on DWARF has been implemented and evaluated [17, 87, 96, 114]. This section discusses the goals that were reached as well as the limitations that were observed with respect to the current DWARF-based Ubitrack implementation.

5.5.1 Current Implementation

We have implemented a demonstration setup that shows the "extended tracker range" setup discussed in section 3.3.1.

Based on several helper libraries, most existing DWARF services handling spatial data were modified such that they implement the Ubitrack sensor API. In addition, viewing components were changed such that they fulfill the query API.

A prototype of the DWARF UMA was implemented, that serves as a proof of concept, although it has to deal with some peculiarities of the current DWARF middleware implementation. The example scenario has a very small number of SR graph nodes and edges. However, this number is representative for current AR tracking setups. As no additional tracking hardware was available, the functionality and performance of the distributed UMA had to be evaluated on synthetic data. For this purpose, a small helper program was implemented that generates random graphs of type $G(n, p)$, i.e. graphs with n vertices and a probability of p for each edge to exist. This graph was then partitioned into several subgraphs that were deployed on distributed UMAs. The UMA network then had to solve several optimal path search requests, the overall time for this task was measured. SR Graphs up to 100 nodes were tested on up to three UMAs, the achieved running times were at a maximum of a second.

The inference component used in the demo application was as simple as possible. A single service had a need for all three tracking devices involved (ART system, static calibration and ARToolkit) and an ability for the aggregated data. This data was processed by the DWARF Viewer that is based on Open Inventor[3]. Figure 5.8 shows the resulting component diagram, including the logical layers. Note that the DWARF service manager has been omitted in the diagram.

In summary, the mapping of the Ubitrack distributed implementation concept was proved to be feasible.

5.5.2 Limitations

The current DWARF-based implementation is far from perfect, although the general concepts serve as a solid basis for further improvements. During the implementation, several issues were discovered:

Dependency on service location: A prerequisite for the distributed Ubitrack implementation concept is that it must be possible to locate information about SR graph nodes in the distributed setup. This *service location* is done via the usual DWARF ability/need mechanisms. Currently, these are based on OpenSLP[4], an implementation of the Service Location Protocol (SLP). SLP

[3] `http://oss.sgi.com/projects/inventor/`
[4] `http://www.openslp.org/`

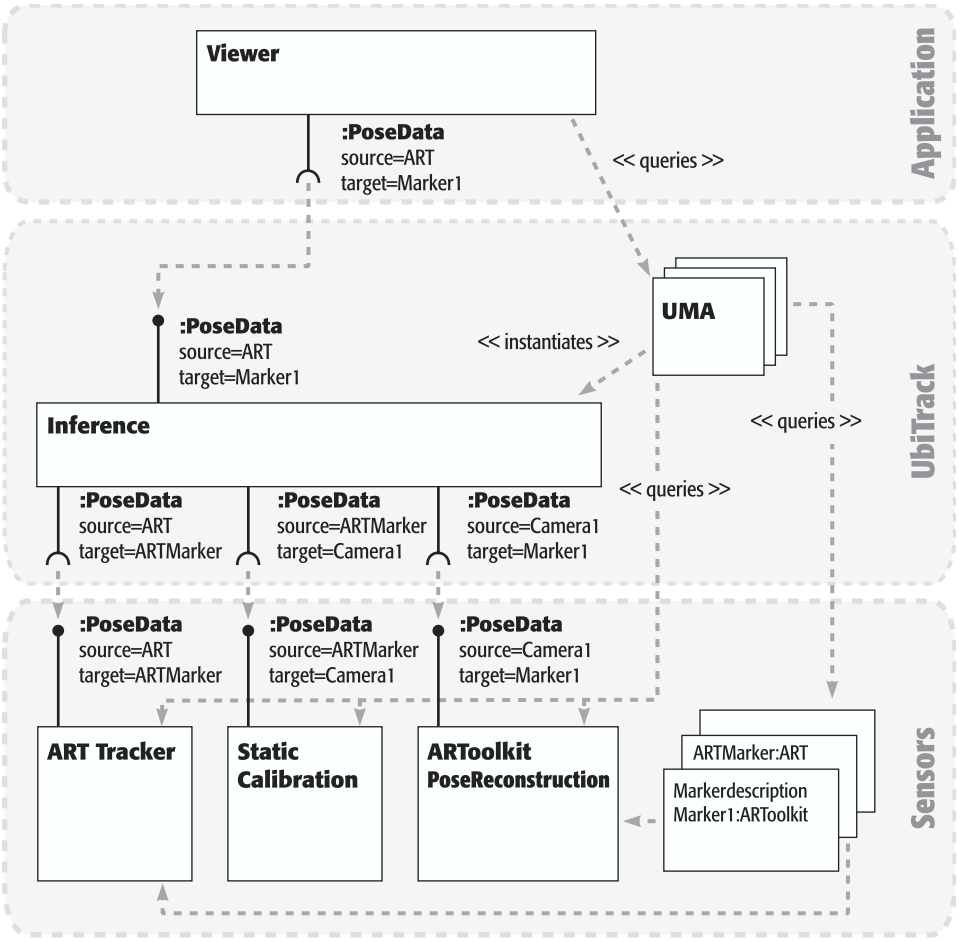


Figure 5.8: UML diagram of the DWARF-based implementation of the extended tracker range scenario.

is based on a polling protocol. Services have to register with a so-called *Service Agent (SA)* or a *Directory Agent (DA)*, the first being reached by multicast messages, the latter by unicast messages. After they have been registered, clients may query a SA or DA for specific services with a given set of attributes. If a matching service can be found, an URL is returned that can be used by the client to set up a communication. With this protocol, it is not possible to be notified as soon as a matching service gets available, instead, constant requests have to be issued. With the large number of service requests necessary for the Ubitrack implementation, it takes quite a long time (potentially up to several seconds) until a service location request can be answered. This time adds up to the time necessary for optimal path computation and setup of inference components.

Restriction to local subnet: In the current DWARF setup, only SLP Service Agents

are used. In consequence, service location functionality is only provided on the local subnet. However, an extension to handle SLP Directory Agents should be easy to implement.

Restricted maximum number of SR subgraph nodes: Due to a bug in the OpenSLP implementation used for DWARF, the maximum number of SR subgraph nodes kept within a single UMA is restricted. Contrary to the SLP standard, OpenSLP limits the size of a packet announcing or requesting a service to roughly 15000 bytes. Thus, the multivalued attributes necessary for the inter-UMA needs and abilities are restricted in size. These attributes contain a list of all local SR graph nodes, consequently, its number is restricted.

Communication overhead with service manager: The DWARF UMA is tightly coupled with the DWARF service manager. The communication overhead between the two processes is significant, it would be better to integrate the UMA's functionality into the service manager.

No optimization of inferences: In the current data flow concept, no global optimizations according to user-defined criteria (e.g. low latency, low energy consumption) is possible; instead, a set of data flow components is set up for every inference that needs to be made, without making use of already existing components. In future revisions of the DWARF Ubitrack implementation, work on distributed multimedia [72] should be applied to the data flow concepts.

CHAPTER 6

Handling Mobile Setups

Overview

This chapter describes a software architecture that allows the seamless integration of multiple Ubitrack setups. Most often, mobile users bring their mobile setups into stationary intelligent environments. Yet, since the Ubitrack architecture is based on the peer-to-peer paradigm, it is also possible to use it to connect two mobile setups.

Both the Ubitrack formalism and the distributed implementations discussed in the preceeding chapters have a global view on a running system and assume every object to be known a priori. This assumptions do not hold true in ubicomp scenarios that consist of applications designed to be assembled at run time. This chapter proposes an architecture supporting the necessary setup prior to run time location estimation.

The architecture is driven by the key requirement that tracking gear has no a priori knowledge of the environment it is operating in. As such, protocols for exchanging configuration data have to be defined for every tracking technology. The chapter starts with an example for the Augmented Reality Toolkit [60], a good representative for the class of fiducial-based feature trackers.

It is explained how the features of the DWARF framework described in section 5.1 are used to make the dynamic configuration work. The key idea is to use DWARF ability attributes and need predicates for a context-aware selection of configuration data. Yet, the configuration architecture can not solve the open research problem of context categorization. This thesis follows Dey's work [35] and discusses the implications of this choice of context model.

For the system to work, all sensors and locatables must have a unique ID and their properties must be known to at least some part of the system. For today's demo setups, this can safely be assumed. However, in more general ubicomp scenarios, we have to cope with new configuration data being generated on the fly, such as descriptions of natural features just detected by a vision-based tracker. Several representative scenarios that show the different cases that can occur are discussed. It will be shown how the configuration architecture needs to be extended to support these scenarios.

6.1 Problem Statement

A key requirement to solve the problem of Ubiquitous Tracking is to handle dynamic changes in the environment. The last chapter discussed how features of the DWARF framework help in handling a changing set of sensors, based on the concept of a time-restricted Ubitrack SR graph.

The situation gets more complex if not only sensors, but rather *mobile setups* worn by users entering an intelligent building have to be integrated dynamically. For example, a user might carry a cell phone that is equipped with a camera. This camera can be used for fiducial tracking [81]. If the user enters an intelligent building, the tracking software on the cell phone usually does not know anything about the properties of the building, i.e. how tracked fiducials should be interpreted. The software can only detect spatial relationships between the camera and fiducials. Software running in the intelligent building can use this data to track the user and probably guide him through the building, assisted by additional stationary trackers.

On the other hand, software running on the user's mobile setup might use services offered by the intelligent building to enhance the user's experience. For example, navigation instructions generated by the building might be displayed in a head-mounted display attached to a computer worn by the user.

The connection of multiple tracking setups is not only useful in scenarios involving intelligent buildings, it can also enhance the user experience if two mobile setups get connected. The *Dynamically Shared Optical Tracking* by Ledermann et al. [70] is a good example how two mobile cameras can collaborate in the green field. However, if we imagine two users meeting spontaneously, there must be some possibilities to exchange the necessary configuration data, such as abstract descriptions of fiducials an optical tracker has to detect.

In consequence, the problem treated in this chapter is to provide a software architecture that allows the reciprocal configuration of location sensing setups joining each other. The architecture should work in a hybrid peer-to-peer fashion to allow a uniform treatment of both the stationary/mobile and mobile/mobile scenarios and to support reusing the DWARF-based Ubitrack implementation discussed in the last chapter.

6.1.1 Configuration of Tracking Hardware

Before defining a software architecture for configuring tracking setups, the necessary configuration data that may be transmitted has to be analyzed.

Analyzing the survey of tracking hardware in section 2.5, the necessity for configuration data can be divided in three classes:

No configuration data: Several tracking systems have an absolute reference frame that has a well-defined interpretation. For example, magnetometers and GPS systems operate on an absolute reference frame. In consequence, no additional configuration data is needed.

A second category of trackers not needing dynamic configuration data consists of devices that are wired and have a fixed setup. For example, commercial mechanic, magnetic field sensing and ultrasonic trackers all need physical linkage between the tracker and the locatable. As such, it is impossible that mobile setups are tracked without attaching special locatables to them.

Base coordinate system and calibration data: Location sensors performing measurements relative to some reference coordinate system might need the spatial relationship of this reference system relative to some other system that is well-defined. For example, a user-worn camera tracking fiducials estimates spatial relationships between itself and the fiducials. If both the user's and the fiducials' location are unknown, the data is of little use. Thus, a base coordinate system is needed to correctly interpret a sensor's readings.

Location sensors estimating *relative* measurements tend to drift over time. Even if a spatial relationship to some well-defined reference is obtained, it must be recalibrated from time to time.

Feature data: This class of tracking devices is the most flexible and currently consists mainly of vision-based trackers. Its general property is that a large number of different and previously unknown features can be observed by the sensor, and that the sensor computes an estimate of the spatial relationship between itself and a subset of these features.

Fiducial-based tracking systems such as ARToolkit or the ART dTrack system need abstract descriptions of artificial markers to work correctly. In the case of ARToolkit, such a description consists of a 32×32 pixel pattern that is to be found inside a black square, for the ARTrack system, the description consists of the geometric relationship of a unique arrangement of retroreflective balls.

A more complicated situation occurs in the case of natural feature based trackers, most of the proposed algorithms (e.g. [63, 84]) require a 3D model of the environment. The configuration data necessary for a mobile tracker brought into a previously unknown environment then consists of such a model.

Additionally, natural feature trackers often have to be initialized with a rough estimate of the camera's initial position. This would also be part of the configuration data.

The first case is simple, and the second case can be solved by mechanisms identical to those described in the last chapter. If we can assume that objects are represented by SR graph nodes with globally unique IDs, the Dwarf UMAs on mobile setups will spontaneously connect with UMAs running in a stationary environment as soon as network coverage and therefore service discovery facilities are available.

The last case is of particular interest for natural feature based trackers, as it allows to use this unobtrusive technology in a much larger area than currently possible. The configuration architecture discussed in this chapter offers a feasible solution.

6.1.2 Configuration Protocols: An Example

A real-world example helps in understanding the structure and necessary amount of feature configuration data. This section discusses which configuration data is needed by ARToolkit and how ARToolkit has been integrated in Dwarf to allow flexible configuration.

ARToolkit is a vision tracking library. It employs artificial features and is most often used as an outside-in tracker. It contains video acquisition code, code for finding black square fiducial markers in the video images, pattern matching code to identify the fiducials, a pose reconstruction algorithm to detect the 6 DOF spatial relationship of the fiducial relative to the camera and code to render OpenGL scenes relative to the fiducial.

ARToolkit needs two types of configuration at startup. First, a *camera configuration file* containing the intrinsic camera parameters. Calibration routines to estimate these are included with the library. Second, a *pattern description file* for each fiducial that is about to be detected. A pattern training library function is included with the library as well. The pattern description consists of a ASCII string with a list of numbers. These numbers represent byte values of a 16×16 sample of the pattern to be identified, with one sample for each RGB color channel. To identify the orientation of the pattern, four such RGB-blocks are stored in the configuration file, representing the pattern turned 0, 90, 180 and 270 degrees.

To modularize ARToolkit, several Dwarf services were designed and implemented:

VideoGrabber: This service takes images from a digital video camera and puts them into a shared memory segment. It offers an ability of type `CameraData` with a connector of the `Shmem` protocol. Currently, two versions of this service

exist, one for IEEE 1394 based cameras and another service for all cameras supported by video for Linux (V4L)[1].

ARTkMarkerDetection: This service takes video images from a shared memory segment and detects fiducial markers in the images. It then sends out the 2D image position of every fiducial it identified, including the fiducial's name and a confidence value indicating the probability that the fiducial matches the given pattern. The service has two needs, `CameraData` with a `Shmem` connector and `ARTkMarkerData` with an `ObjrefExporter` connector, and an ability of type `artkFrameMarkers` with a `PushSupplier` connector. The `ARTkMarkerData` need is used to offer a configuration interface.

ARTkPoseReconstruct: This service takes the 2D image position of fiducials and reconstructs their 6 DOF pose relative to the camera. Consequently, it has a need of type `artkFrameMarkers`. At startup, it parses its list of abilities for those of type `PoseData`. For every ability found, it evaluates its attribute `MarkerName` and configures itself in a way that the pose of a fiducial of identical name gets reconstructed out of its 2D image position. The `PoseData` resulting from this process can be used by existing DWARF services (e.g. viewing components) as any other tracker output.

ARTkMarkerConfigurator: This service offers a GUI for configuring the *ARTkMarkerDetection* service. Thus, it has an ability of type `ARTkMarkerData` with a `ObjrefImporter` connector. Figure 6.1 shows a screenshot of the service. The top menu items are used to add or remove markers from the list, and the *loaded/activated* check boxes in the list are used to send corresponding commands potentially including a binary representation of the pattern to be loaded to a connected *ARTkMarkerDetection* service.

In a typical mobile user scenario, the mobile setup consists of an instance of the *VideoGrabber* service and an instance of the *ARTkMarkerDetection* service. Both have to run on the same machine as they communicate via shared memory. In the stationary environment, an instance of the *ARTkPoseReconstruct* service receives the `artkFrameMarkers` events from the mobile marker detection service and provides the resulting 6 DOF spatial information to other services, both in the environment and on the mobile setup. The advantage of the pose reconstruction running in the stationary environment is that it can be configured beforehand to the specific setting. As a result, it could also take into account environmental conditions like knowledge about walls using dead reckoning algorithms. The necessary configuration data consists of two parts:

Pattern descriptions: The stationary environment must transmit descriptions of fiducial patterns to the marker detection service on the mobile setup.

[1] `http://www.exploits.org/v4l/`

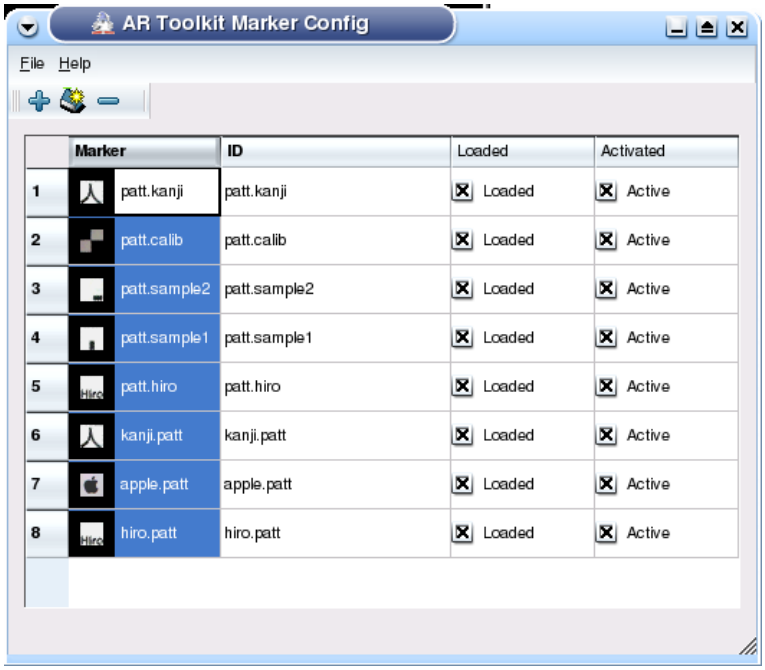


Figure 6.1: Screenshot of the ARTkMarkerConfigurator DWARF service.

Camera parameters: The mobile video grabber service must transmit the intrinsic parameters of its video camera to the pose reconstruction service in the environment.

The remainder of this chapter discusses flexible and reusable solutions for providing this data.

6.2 Distributed Spatial Configuration Architecture

The proposed architecture is based on the DWARF framework and uses the dynamic features discussed in section 5.1.3. First, the key requirements for the architecture are derived, then a DWARF-based solution is discussed.

6.2.1 Requirements

The mobile ARToolkit tracker scenario has illustrated the typical usage pattern of the desired configuration architecture. The following requirements can be derived from it:

Spatial organization of configuration data: Configuration data organized according to its spatial properties must be supported. For example, the descriptions

of all features in a building must be split into spatial entities such as individual rooms. This requirement ensures manageability of configuration data and allows to limit the number of features a sensor must distinguish concurrently. Note that it is also possible to create a spatial hierarchy, e.g. the first floor contains rooms 101 through 110. The feature sets for entities in higher hierarchical levels may be different from the union of the lower levels' entries. For example, the first floor's feature set may consist of simple to detect features that can be used to determine which room the camera is in, once this is done, the fiducial tracker can be reconfigured such that it knows the features of this particular room. Note that the commonly used scene graph representation of objects does not solve this problem: a scene graph only facilitates the structuring of information, but it is still possible to organize it in a way that has nothing to do with the spatial organization. Spatial organization of configuration data is the key difference of the discussed distributed architecture to existing systems such as the Nexus platform [89], which require a central data storage component.

Transparent access to configuration data: A component that needs configuration data should not have to care about where it gets this data from. This requirement ensures that setups can connect to others without any a priori knowledge about the connecting partner. For example, a user can take his mobile setup into buildings he has never been before.

Spatial indexing: It must be possible to integrate spatial indexing, i.e. naming of spatial regions. Up to now, the spatial relationships described in this thesis have been of pure geometric nature – coordinates given in some reference frame. To allow a semantically rich description of spatial regions, it is extremely helpful to name certain regions, for example, "Room 01.07.057" or "Campus Garching". How such knowledge is derived out of sensor data is beyond the scope of this work, a simple method would consist of RFID tag readers at a door identifying users entering or leaving a room, more complex methods have been discussed in literature [3, 45, 83].

6.2.2 Prerequisites

The proposed architecture is based on some assumptions on the environment in which it should be deployed:

Network coverage: An almost permanent network coverage, preferably wireless, is assumed. Mobile setups connect automatically or manually to the network upon a user's request, e.g. via a cell phone.

Availability of spatially indexed data: The architecture operates on string-based comparisons of spatial entities. In consequence, the derivation of these entities' names out of raw sensor data must be available in the environment.

Configuration data set up to match spatial entities: The authoring process for configuration data must take into account the spatial entities derived by the indexing process. Each piece of configuration data must be associated with a single or some spatial entities.

Standard protocols for configuration data: For every device class that should be configured, a standardized protocol must be defined. In the example above, the protocol for the ARToolkit based fiducial tracker consists of remote method calls to load or activate pattern data.

Rapid service location: The architecture is based on service location protocols. After a change of the spatial entity is detected, new DWARF services have to be located and connected to apply the configuration change. In consequence, the service location has to be happen fast enough to make the configuration architecture work.

6.2.3 Architecture

The key ideas of the proposed architecture rely on the concepts of *attributes* of *abilities* and *predicates* of *needs* within DWARF. Remember that attributes are name/value pairs of strings, and predicates are boolean expressions limiting the choice of available abilities to those with attributes fulfilling the expression.

Every service is associated with a spatial entity. For example, a *ARTkMarkerDetection* service running on a mobile user-worn setup is always associated with the spatial entity the user is currently in. The spatial entity of a service gets expressed by an attribute with name `Room`, e.g. `Room=Hallway`.

To get notified of changes in its spatial entity, every service has a special need of type `ContextSwitch` (location is just a special case of context). Spatial indexing services offer corresponding abilities of type `ContextSwitch`. Whenever they detect a change in the spatial entity, they send an event indicating the new entity's name to the event channel. For example, an indexing service observing a user in the room *ARLab* may detect that the user just leaves the lab to the room *Hallway*. It then sends an event with this observation to the interested services.

Services that need configuration data use the knowledge about their spatial entities to add predicates to their needs for configuration data. For example, the *ARTkMarkerDetection* service's need for `ARTkMarkerData` might initially be enriched by the predicate `(Room=ARLab)`. Upon receiving an event telling the service that its new spatial entity is the hallway, the need's predicate is changed to `(Room=Hallway)`.

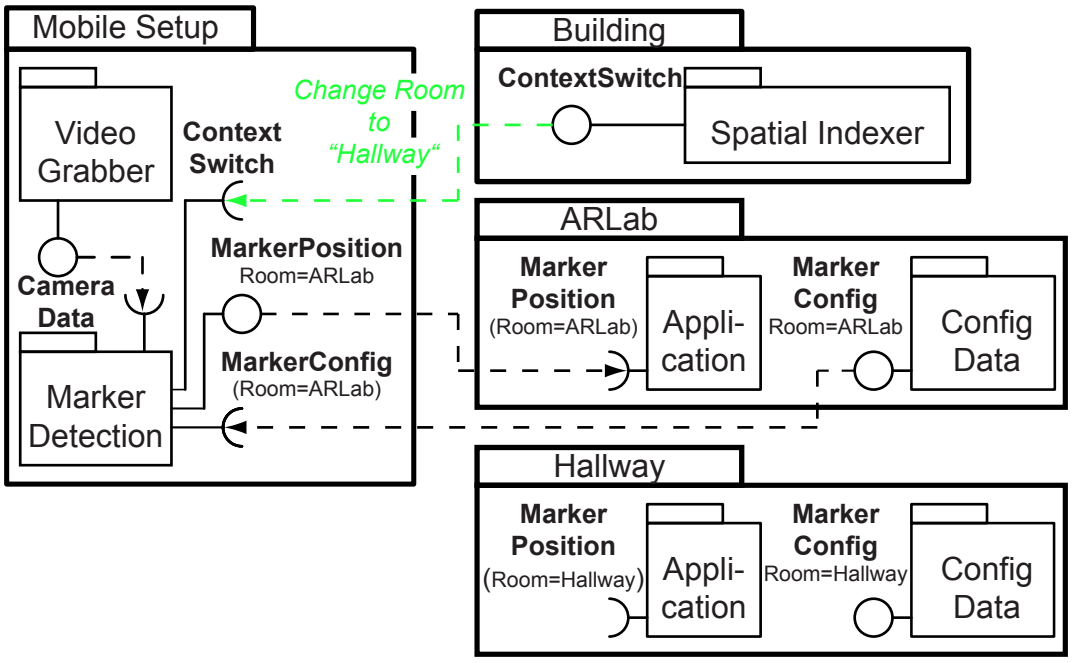


Figure 6.2: Example setup. The spatial indexer notices that the user carrying the mobile setup leaves the AR lab and goes to the hallway. It sends a corresponding event to all interested services on the mobile setup.

As configuration data is organized spatially, the task of its distribution can be split into several abilities. Each of these abilities is enriched by a potentially multivalued attribute of name `Room` expressing the spatial entity the configuration data is for. In our example, the configuration service has two abilities of type `ARTkMarkerData`, one with the attribute `Room=ARLab`, the other with the attribute `Room=Hallway`.

The DWARF service manager detects whenever a service's need changes its predicate such that it does not match a currently set up connection. It then disconnects the two services and tries to find a new matching partner. If it is found, a new connection is set up. This search process happens transparently to the participating services and depends on underlying service location mechanisms. The services can get notified of (dis)connections if they implement the `SvcSession` interface via the methods `newSession(..)` and `endSession(...)`. In our example, we assume initially the *ARTkMarkerDetection* service being connected with the *MarkerConfiguration* service of the AR lab. When the user leaves the lab, the `ARTkMarkerData` need's predicate gets changed to `(Room=Hallway)`, and the service manager consequently disconnects the service from the lab's *MarkerConfiguration* service and connects it to the hallway's configuration service.

It is the participating service's responsibility to perform suitable actions upon

(dis)connection. In our example, the detection service would notice its disconnection from the lab's configuration service by a call to `endSession(...)`. It should then unload all pattern descriptions it received from this service. On connection to the hallway's configuration service, the latter would also get notified via `newSession(...)`, and should trigger an upload of all hallway pattern data using adequate method calls.

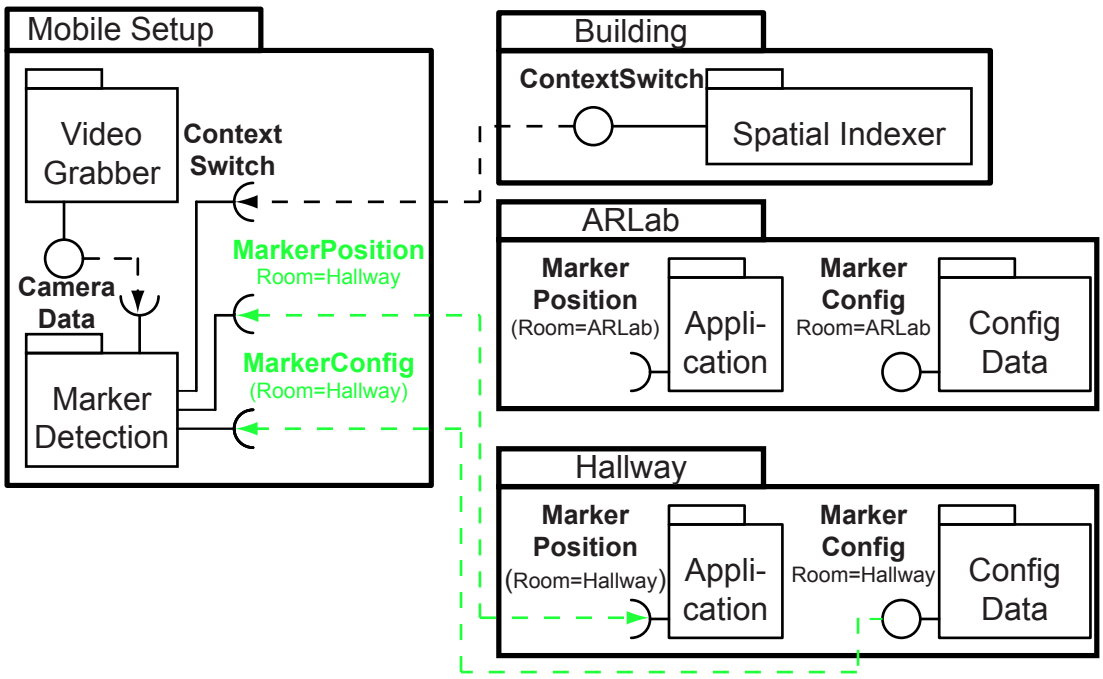


Figure 6.3: Example setup after configuration change. The marker detection service changed its ability attributes and need predicates such that it gets a new configuration from the hallway's corresponding service and sends its data to another application.

6.2.4 Possible Extensions

The basic principle of location-dependent connection of configuration components to services running on mobile setups can be extended easily in several directions.

Location-aware choice of spatial indexer: The spatial indexer itself can be treated in a similar way as the configuration component by adding a `Room` attribute and extending the `ContextSwitch` need by a corresponding predicate. This mechanism can be used to provide several spatial indexers simultaneously, either to increase the scalability or to facilitate the management of these components.

Applications on mobile setup: In the example above, the application resided in the stationary environment. However, there may also be applications running on the mobile setup itself. If, for example, we assume the mobile setup to be equipped by a head-mounted display and a matching viewer component, it can be used to visually augment the user's view of the environment. The configuration of the viewing component would then come both from the stationary environment (e.g. scene descriptions of virtual objects) and the mobile setup itself (e.g. calibration parameters for optical see-through operation). In addition, the mobile setup can make use of sensor data available in the environment.

Configure stationary environment by mobile setup: The roles of the mobile and stationary parts of the system can be exchanged. For example, a mobile setup might contain some data on natural (e.g. hair color) or artifical (e.g. an ARToolkit marker on a backpack, or hair color) features a stationary tracking system might use to identify the user carrying the setup.

Reciprocal configuration of mobile setups: As the proposed architecture works in a hybrid peer-to-peer fashion with a peer on every network node, it can also be used for two mobile setups being connected on the green field. For example, the concepts of *Dynamically Shared Optical Tracking* [70] can be used to increase the tracking range and/or accuracy if another mobile setup is around.

6.2.5 Results

In addition to the demo application described in chapter 7, the mobile configuration architecture has been implemented and evaluated within the *ARCHIE (Augmented Reality Collaborative Home Improvement Environment)*[2] project. ARToolkit was used as a tracking technology, it was split into the components *VideoGrabber*, *ARTkMarkerDetection*, *ARTkMarkerConfigurator* and *ARTkPoseReconstruct* as described above. The application consisted of a simple *Speaker* service telling the user some information about the room he was currently in. All involved services were distributed on two mobile computers connected wirelessly with each other and the environment.

Using ARToolkit for Spatial Indexing. Within the ARCHIE scenario, ARToolkit itself is used for spatial indexing [126, 127]. The general idea is to attach well-known *transitional markers* to the boundaries of spatial entities, e.g. doors. The *ARTkMarkerDetection* is connected to the spatial indexer and sends the current set of observed markers. As soon as a set of transitional markers is detected in a

[2] `http://wwwbruegge.in.tum.de/DWARF/ProjectArchie`

way that allows to conclude a change of spatial entity, e.g. that the user left the current room, a `ContextSwitch` event is sent to interested components.

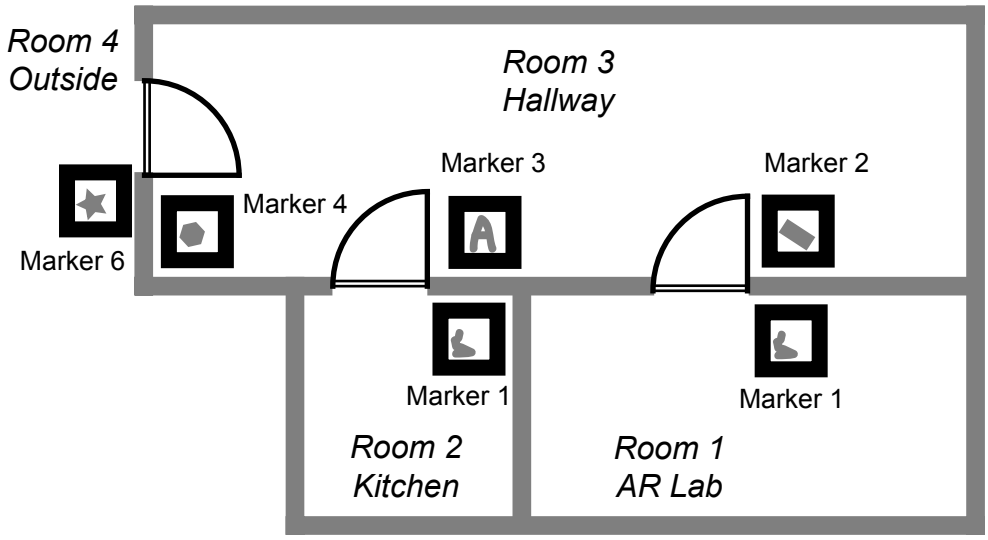


Figure 6.4: Physical setup of ARCHIE scenario. Note that the ARToolkit markers used for spatial indexing can be reused, the outgoing marker both in the ARLab and the kitchen are identical.

Figure 6.4 shows the physical setup of the ARCHIE scenario, consisting of four spatial entities. In this case, the spatial indexer consisted of a simple state machine that was triggered by *MarkerPosition* events from the *ARTkMarkerDetection* service. The state transition diagram is shown in figure 6.5.

Discussion. The configuration architecture was successfully applied to the given scenario. All components reconnected at runtime, and the "start on demand" feature of the DWARF middleware was used to automatically start and stop configuration services depending on other service's needs.

Using the ARToolkit as basis for spatial indexing was not without problems. In general, the toolkit's detection accuracy is too low to use it to trigger locational state changes for the mobile setup. If the spatial indexer misconfigures the mobile setup, recovery gets a major problem. In consequence, for real-world setups other, more reliable, spatial indexing techniques should be used, for the *Ubiquitous SHEEP* demo application described in the next chapter, we decided on using iButtons to explicitly trigger locational state changes.

Nevertheless, the ARCHIE scenario demonstrated the high flexibility of the proposed configuration architecture, allowing mobile and stationary components to use each others' capabilities in an ad-hoc fashion.

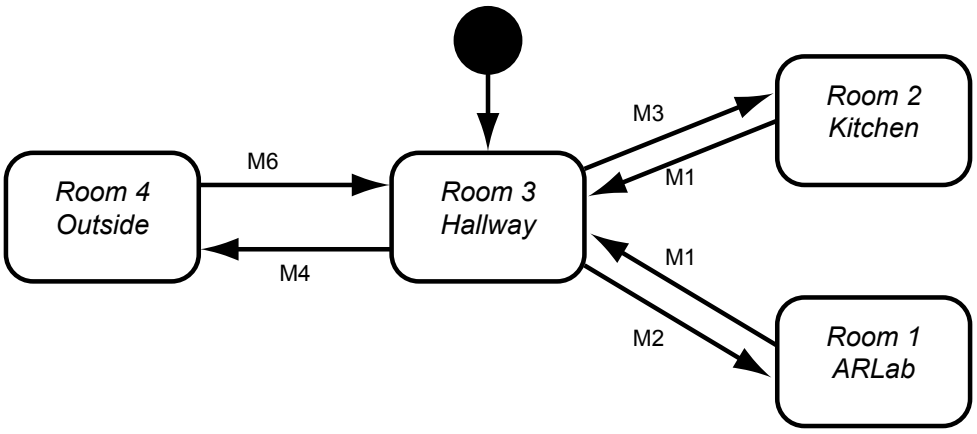


Figure 6.5: State transition diagram of ARCHIE spatial indexer.

The dependency on the service location facilities of the underlying DWARF middleware is a major issue in the current implementation. The slow location speed of the currently used SLP implementation (see section 5.5.2 for details) slows down the reconfiguration of the mobile setup significantly. However, the service location of DWARF can be made sufficiently fast by simply exchanging the protocol [77].

6.3 Integrating Contextual Information in Dwarf

The spatial configuration data just described can be extended to allow for the integration of contextual information in DWARF systems. This section describes the underlying theory, how the architecture has to be extended and discusses the limitations of this approach.

6.3.1 Categorization of Context and Context-Awareness

Dey and Abowd [35] give an excellent categorization of context and context awareness. This section summarizes their findings and shows how the necessary information can be derived in a typical DWARF-based ubicomp scenario.

Contextual information can be divided into *activity*, *identity*, *location* and *time*. For a typical ubicomp scenario, the activity of a user can often be associated with a particular application he is currently executing, the user's identity could be stored on the mobile setup and the last chapters discussed how to derive the user's or some objects' current location. In addition to these simple rules of thumb, more complicated machine learning techniques and software architectures such as the context toolkit [34] can be used to derive the current context state within an application.

Context-aware applications can either *present* information and services to a user,

execute a service automatically or *tag* context to information for later retrieval. As an example, in typical ubicomp scenarios with mobile users coming to stationary environments, information presentation might consist of navigation hints that guide the user to a meeting room, automatic service execution might be an automatic recording of the meeting once the user reaches the room, and tagging might consist of associating parts of the recorded meeting with the current activities of the meeting's participants. The remainder of this section discusses how all these facilities can be implemented with an extended configuration architecture.

6.3.2 Integrating Context in Dwarf

Dey's context categorization maps nicely to the DWARF concept of enriching abilities with attributes and of fulfilling needs conditionally using predicates. Four attributes get associated with each context-aware DWARF service: `Activity`, `User`, `Room`, and `Time`. If another service wants to access information depending on the current context, it has to add predicates to its needs that specify the relevant context criteria. For example, application *BikeRace* might be running on *Joe's* mobile setup. The application is interested in all relevant virtual content, consequently, both predicates `(Activity=BikeRace)` and `(User=Joe)` must be fulfilled for its `VirtualContent` need. With this set of predicates, the application gets all available content, regardless of Joe's current spatial location. If it is desirable to filter the content based on the location, a predicate like `(Room=AlpinePasture)` must be added.

The listed types of context-aware applications can also be supported by DWARF mechanisms. The *BikeRace* application is an example of information presentation, similar presentation tasks work analogously.

Automatic service execution can be implemented using the start on demand feature that automatically starts a service as soon as some of its abilities are requested and all necessary needs are fulfilled, and stops it if either the abilities are not needed anymore or some necessary need stops being fulfilled. For example, a *MeetingRecorder* application might have a need of type `UserDescription` with the predicate `(Room=MeetingRoom)`. We assume every user to have his own mobile setup with some service running on it which has an ability of type `UserDescription` and gives the user's name, contact information and a picture. If the minimum number of ability instances for this need is set to five, the *MeetingRecorder* application is automatically started as soon as at least five people with a suitable mobile setup are in the meeting room. As soon as the number of people drops below five, the application is stopped automatically.

Finally, tagging information with contextual information can be implemented by a service having a service with a need for `ContextSwitch`. This service then gets notified of changes in the contextual state, allowing it to tag data.

6.3.3 Limitations

Although the context architecture just described seems conceptually easy, it is not without problems:

Clear definition of context terms: To make the approach work, a stringent definition of contextual terms is necessary. Only if different applications have a clear understanding of the semantics of certain attributes is it possible to let them interact. Unique names have to be assigned, for example, the attribute `User=Joe` might indicate completely different Joes.

Manual generation of context classification: It is not completely clear how automatic unsupervised context estimation (e.g. [66]) can be integrated. Using the approach of separating context estimation to a distinct component allows to encapsulate context generation, however, the simple name/value scheme of the contextual attributes restricts the flexibility of the output of context estimators.

No implicit contextual hierarchy: The proposed contextual attribution scheme is flat. It is not possible to express hierarchic relationships like "Room ARLab is on the 2nd floor of building B" or "User Joe is part of the bike race team". Such knowledge must be modeled externally, there are no standard ways to incorporate hierarchies.

It seems promising to put more work into the spatial configuration architecture, as all of the problems just described occur for the locational part of contextual information as well. However, they might be easier to solve than for other context categories. 3D spatial information is inherently hierarchical, humans are used to think of it in containment relationships. There exists also a wealth of related work on spatial indexing (e.g. [3, 45, 83]) that can be used to gain insights on how naming of spatial entities can be generalized.

6.4 Identifying Objects

Up to now, it was assumed that all objects within a Ubitrack environment have a unique name and can therefore be identified by all components of the system. In practical setups, this assumption is unrealistic: mobile users come into stationary environments to which they have never been to and still want to connect to the services offered by that environment. How should the mobile and the stationary setup obtain knowledge about each other? Some *bootstrapping* mechanisms have to be applied in order to allow the identification of previous unknown objects in a Ubitrack system. This section discusses some ideas on how such mechanisms should be structured in order to allow the setup of sensor networks that contain unidentified objects.

6.4.1 Problem Definition

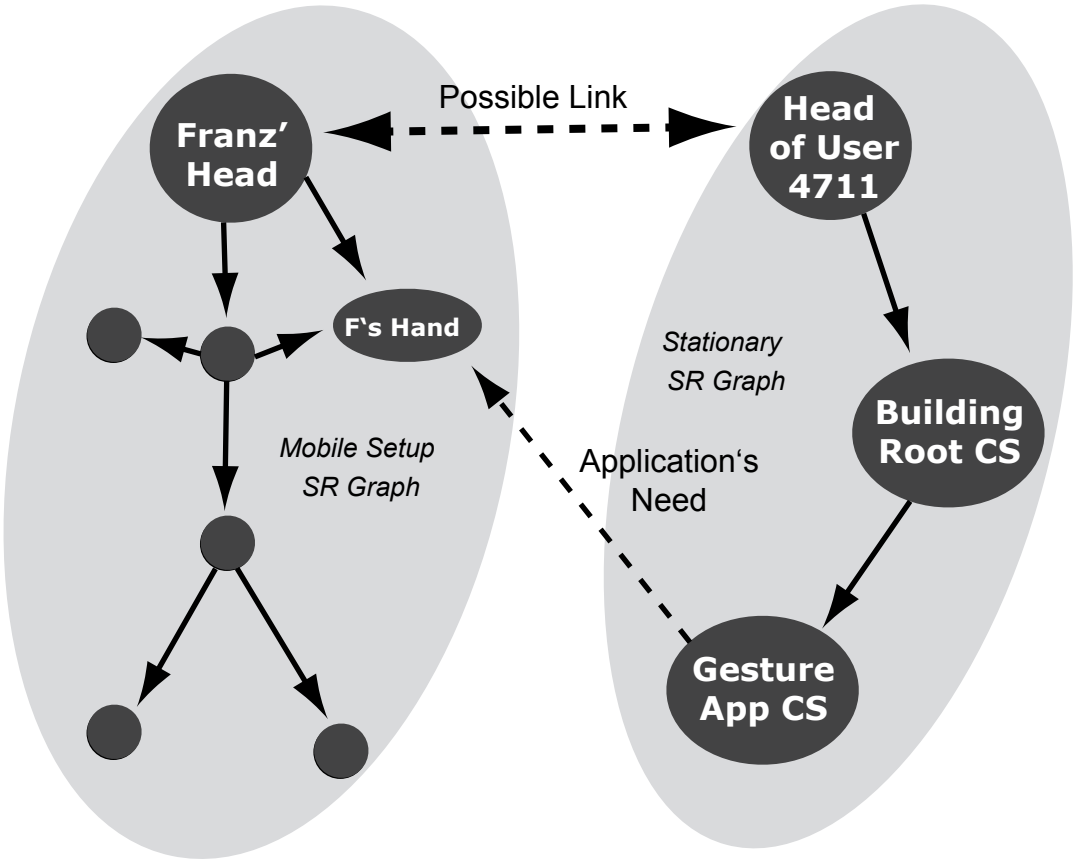


Figure 6.6: Two SR graphs of a mobile setup and a stationary environment. The graphs should be connected by adding an edge between the nodes *Franz' Head* and *Head of User 4711*.

The Ubitrack formalism, the distributed implementation concept and the implementation based on DWARF all assume that every object within a SR graph has a unique ID. This is necessary to allow the connection of two SR graphs. As an example, user *Franz* carries a mobile setup with several location sensors. These sensors are networked and form a Ubitrack SR graph. Some node in this graph is called *Franz' Head*, and we assume that the mobile sensors can estimate the position of Franz' hand relative to his head, i.e. we are given some inference edge in the SR graph from the node *Franz' Head* to the node *Franz' Hand*. If Franz now enters a building, some video camera mounted on the ceiling (and integrated into the building's Ubitrack system) might detect his head's position relative to

the building. The software analyzing the video stream gives every detected head a sequence number, in consequence, it may assign the ID *Head of User 4711* to Franz' head. An application running in the building is interested in detecting hand gestures and associating them with some actions. In consequence, we are given the two SR graphs depicted in figure 6.6, with the application's needed spatial relationship indicated by the dashed line.

In the example, the sensor network bootstrapping problem consists of finding out that the objects *Franz' Head* of the mobile SR graph and *Head of User 4711* are either identical or have a static spatial relationship. In general, the problem is to find a spatial relationship between two nodes located in different SR graphs to be merged. After this finding, an edge associated with an identity or static relationship can be added between the two nodes, thus merging the two SR graphs.

6.4.2 Transmitting Object Descriptions

In most cases, the spatial configuration architecture described in section 6.2 can be used to prevent the object identification problem. If a sensor can be configured with data from a remote system, the remote system can also transmit the object's ID. In consequence, a spatial relationship between the new object and the mobile sensor can be derived, thus merging two SR graphs.

If a stationary locatable is to be identified by a mobile sensor, e.g. a camera, the setup of figure 6.7 results.

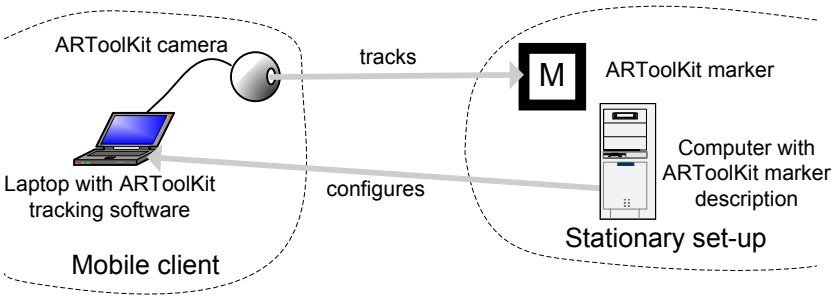


Figure 6.7: A stationary locatable is to be identified by a mobile camera. Thus, the stationary environment has to transmit information about the locatable to the mobile setup. (Taken from [114].)

Figure 6.8 depicts a similar situation, with the roles of the mobile and stationary setups exchanged. The locatable description might consist of a pattern description as that of ARToolkit described in section 6.1.2, but it could also be some abstract description such as a histogram of hair color distribution for the ceiling-mounted camera of the example above.

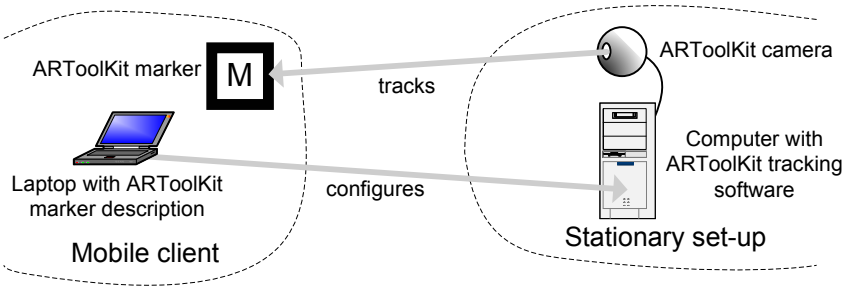


Figure 6.8: A mobile locatable is to be identified by a stationary camera. Thus, the mobile setup has to transmit information about the locatable to the stationary environment. (Taken from [114].)

Bootstrapping Configurations. Imagine a vision sensor of a mobile setup should be connected with the stationary environment of a building using the spatial configuration architecture. The building might hold thousands of feature descriptions. If all are sent to the vision sensor at once, it is very likely that it will misclassify many observations. In consequence, a largely reduced set of feature descriptions should be used.

The feature set can be reduced by introducing a hierarchy of spatial entities, as depicted in figure 6.9. The hierarchy can be presented as a tree. With every node, a set of feature descriptions corresponding to the level of detail of the current spatial entity is associated. For example, if a mobile user starts up his vision sensor, it is initially in the spatial entity *Nowhere*. Only local features necessary e.g. for estimating the spatial relationship of the user's hand relative to his head are loaded into the sensor.

As soon as the user comes to the spatial entity *FMI Building*, a set of features sufficient for coarse navigation (e.g. descriptions of features distinguishing different floors) is loaded. This process goes on until the leaf nodes of the spatial entity tree, where all features available for high-accuracy local sensing are loaded.

6.4.3 Anonymous Objects

The situation gets much more complicated if object descriptions can not be transmitted. This might be the case for natural feature trackers that detect trackable features on the fly without prior knowledge. The resulting object nodes in the SR graph are *anonymous*, as they do not have any semantics associated although they might have a name like *Object4711*.

Strasser [114] reports promising experiments on using frequency analysis of angular and positional velocity of objects to detect whether two objects in different reference coordinate frames have a static spatial relationship. If one of these objects is anonymous, it can consequently be integrated into an existing SR graph.

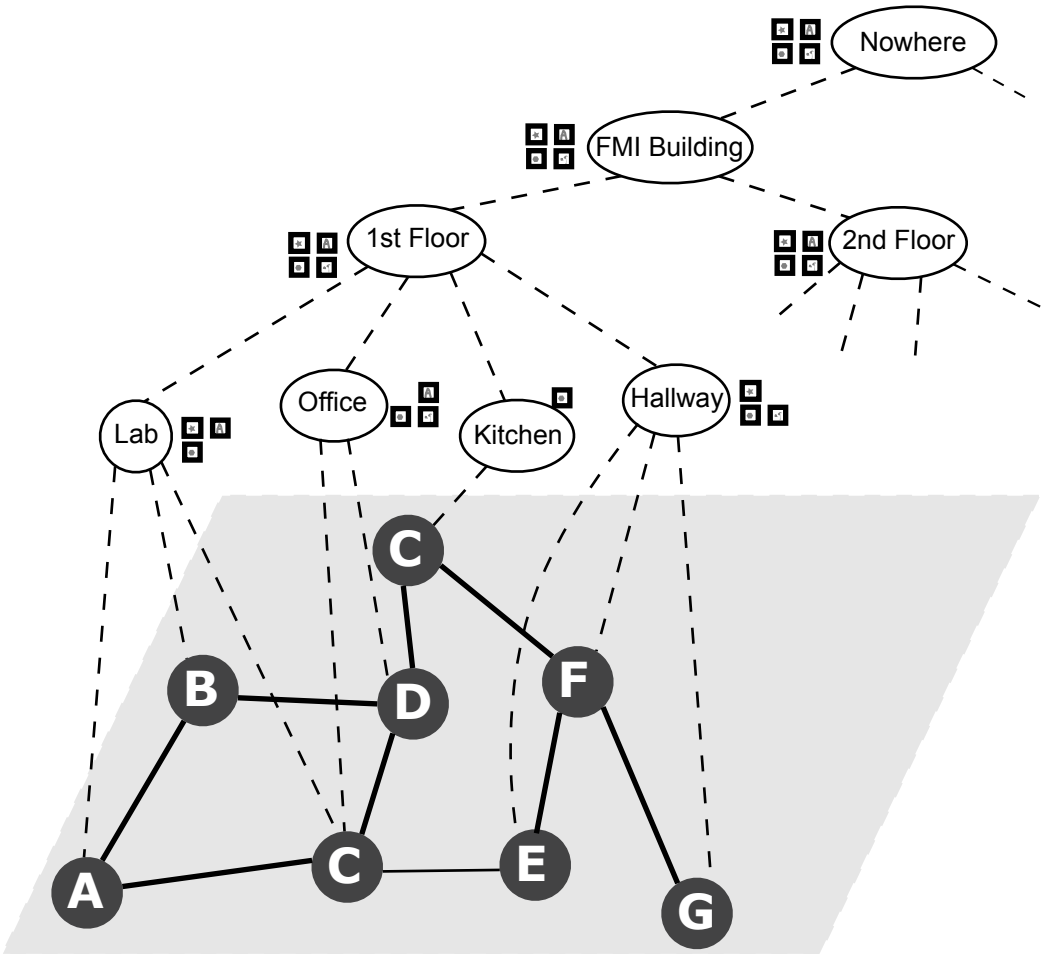


Figure 6.9: Hierarchical reduction of configuration data. At every hierarchical step, new sets of object descriptions are used.

This work is ongoing and beyond the scope of this thesis. For the time being it is assumed that some part of the distributed sensor system has knowledge about every object whose spatial relationship to some other object can be measured.

CHAPTER 7

Ubiquitous SHEEP: A Demo Application

Overview

This chapter discusses *Ubiquitous SHEEP*, an application that demonstrates all concepts of the previous chapters in an integrated prototypical system.

A user with a mobile setup is roaming some rooms in our computer science building, making use of location sensors mounted stationarily in individual rooms and mobile location sensors he is carrying with him. All those tracking devices are configured dynamically and in a context-aware fashion by the running applications. Their output is delivered to the Ubitrack layer and combined automatically such that full use of all available information can be made.

A detailed system design is presented, including in-depth descriptions of the subsystems involved in the demo: tracking, presentation, interaction, virtual sheep simulation and context. The tracking subsystem employs the Ubitrack formalism described in chapter 3 and several parts of the DWARF-based Ubitrack implementation of chapter 5. All mobile parts of the demo setup are configured according to the distributed spatial configuration architecture described in chapter 6.

The chapter concludes with a discussion of the resulting system, its performance and lessons learned from implementing the demonstration setup.

7.1 Scenario

The scenario of the *Ubiquitous SHEEP* demo is based on the original version of the Shared Environment Entertainment Pasture (SHEEP) [78] described in section 3.3.3.

Game-Playing Scenario. A user is equipped with a mobile setup consisting of a laptop with a video camera attached in a way that allows video see through operation. The video stream is also used for location sensing, as are an inertial sensor and an iButton reader. The user carries a trackable paddle to interact with the system.

On system startup, the user has to give the mobile setup some hint about his current coarse location. He does this by briefly touching an iButton attached to some door in his vicinity. The mobile setup then configures according to the current room the user is in.

In the *ARLab*, a projection table displays a pasture with a herd of virtual sheep running around. A tangible real plastic sheep is tracked and can be used to control the herd's behavior. With a pointing device and a set of speech commands, the user can create or remove sheep on the pasture. The user gets a video see through AR presentation of the pasture and the real and virtual sheep on his laptop. He can pick up a sheep from the table using his paddle; the sheep is then removed from the herd and sitting on the paddle.

With the sheep on the paddle, the user can leave the ARLab and go to an *Office* via the *Hallway*. In the hallway, he can color the sheep by moving the sheep on the paddle into some bucket filled with virtual red dye.

In the office, another herd of virtual sheep is running around on a monitor. The user can drop the red sheep on his paddle to the monitor. The red sheep will then automatically join the monitor's herd.

Sheep from the monitor herd can also be carried to the projection table herd. Furthermore, sheep can be picked up from a herd, recolored and placed back to the same herd again.

Application Areas of Scenario. Although the Ubiquitous SHEEP scenario is of mere academic nature, it can be modified slightly such that real-world applications can be realized. For example, a mechanic could carry a similar mobile setup when he should maintain some machinery. The configuration architecture could then be used to transmit maintenance information. The last exchange time of some parts or part numbers could be transmitted directly from the object under maintenance.

In addition, the same concepts that are used to carry around and color virtual sheep can be used to control some machinery parts that are exchanged by the mechanic and taken away for overhaul.

7.2 Sensing Technology

To make Ubiquitous SHEEP work, multiple diverse sensors are employed. This section describes which objects are tracked by what sensors and how the sensors are configured.

ART dTrack: The AR Lab is equipped with an ART dTrack[1] high-precision 6 DOF absolute pose system. It tracks the tangible sheep, the user's laptop and a pointing device used both for calibrating the system and inserting or removing sheep from the herd. The DWARF *ARTTracker* service encapsulating this sensors is configured statically to track the tangible sheep and the magic wand. The mobile laptop is also preconfigured, however, the configuration architecture described in the last chapter can be employed to transmit the marker information from the laptop to the stationary equipment at the moment the user enters the AR Lab.

ARToolkit: The video stream of the laptop-mounted camera is analyzed by the DWARF-adapted ARToolkit software[2] described in section 6.1.2. The detection service's need `ARTkMarkerData` is configured in a context-aware fashion, depending on the user's current spatial entity:

- On startup, only the description of the ARToolkit marker attached to the user's paddle is loaded and the paddle's pose is always registered relative to the video camera on the laptop.
- In the AR Lab, no additional markers are necessary.
- In the hallway, the description of the marker of the virtual dye bucket are loaded.
- In the office, the description of a marker attached to the monitor with the herd on it are loaded.

XSens MT9: An XSens MT9[3] 3 DOF absolute orientation tracker is attached to the laptop next to the video camera. This inertial tracker's readings are fused with the data delivered by the ARToolkit tracker to enhance the quality of pose estimations of objects that are fixed relative to the world reference frame, i.e. the virtual dye bucket and the monitor.

iButton Reader: An iButton Reader from Dallas Semiconductor[4] is used to tell the system whenever the user changes his spatial entity. For this purpose, every relevant door is equipped with two iButtons, one on each side, that the user has to touch briefly with a handheld probe. The reader gets the iButton's ID and can then deduce the new spatial entity of the user. It is clear that this is a somewhat inconvenient albeit simple process when interacting with a ubicomp environment. However, the focus of this thesis and consequently the Ubiquitous SHEEP demo is not on methods for spatial indexing but rather

[1] `http://www.ar-tracking.de/`
[2] `http://www.hitl.washington.edu/artoolkit/`
[3] `http://www.xsens.com/mt9.htm`
[4] `http://www.ibutton.com/`

on providing a software architecture for spatially indexed environments. As such, the output of a spatial indexer can be emulated in a very natural fashion with the iButtons.

7.3 Ubiquitous SHEEP System Architecture

Ubiquitous SHEEP is an extension of the original SHEEP demonstration. In consequence, several DWARF services can be reused, however, they have to be modified such that they cope with the new ubicomp environment.

The original SHEEP demo focused on interaction possibilities within AR environments, Ubiquitous SHEEP focusses on the tracking abstraction and distributed configuration issues of this thesis. In consequence, the system architecture of the original SHEEP demo had to be modified heavily. Ubiquitous SHEEP can be divided into subsystems with the following responsibilities:

Tracking: This is a Ubitrack abstraction from the sensors used. Its purpose is to deliver all necessary spatial relationships to the other subsystems.

Presentation: This subsystem is responsible for displaying all necessary information to the user of the system. In summary, pastures with virtual sheep both in the AR Lab and the Office locations, a virtual sheep sitting on the user's paddle and a bucket full of virtual dye have to be displayed.

Interaction: This subsystem is responsible for detecting interaction requests of the user. It consists of a collision detection service that triggers events based on physical proximity of real and/or virtual objects and a reused Petri-net based *User Interface Controller* allowing the user to combine pointing gestures with speech commands for inserting or removing sheep from a pasture.

Virtual sheep simulation: This subsystem forms the core of the application, providing virtual sheep that form a herd and communicate their appearance to the presentation subsystem.

Context subsystem: It is responsible for estimating and providing the relevant contextual information. Based on this information, the services on the user's mobile setup and virtual sheep being carried around change their needs for configuration data and consequently adapt their behavior in a context-aware fashion.

The remainder of this section details the specific subsystems.

7.3.1 Tracking Subsystem

Section 3.3.3 describes the major drawbacks of the original demo's tracking subsystem and most of the modifications to make it usable within Ubiquitous SHEEP. This section details the SR graphs of the three different spatial entities the Ubiquitous SHEEP demo operates in.

AR Lab. The user's paddle is tracked by an ARToolkit based software analyzing the video stream. Both the video camera and an ART dTrack locatable are attached rigidly to the laptop. In consequence, the tracking data from the ART system can be used to get the laptop's position relative to the projection table and can be combined with the ARToolkit data to estimate the pose of the paddle relative to the projection table and to virtual sheep. The 3 DOF orientation measurements of the XSens MT9 device are used to enhance the orientational part of the spatial relationship between the projection table and the laptop.

To detect when the user wants to pick up a sheep, the system has to check for collisions of the paddle with any virtual sheep. If the user wants to drop a sheep from his paddle onto the table, a collision between the table and the paddle has to be detected. Figure 7.1 shows the resulting SR graph.

Office. The situation is slightly different for the monitor-based version of SHEEP running in the *Office* location. Now the ARToolkit based tracker determines both the pose of the laptop relative to the monitor and the paddle's pose. Neither a tangible sheep nor a magic wand exist, although they might be added easily. Again, the 3 DOF orientation measurements of the XSens MT9 device are used to enhance the orientational part of the spatial relationship between the world-fixed monitor and the laptop.

Figure 7.2 shows the resulting SR graph. It includes the necessary inferences for estimating the virtual camera viewpoint of the scene graph running rendered on the laptop and for detecting collisions between the paddle and sheep or the pasture.

Hallway. In the hallway, the ARToolkit based tracker estimates the pose of the dye bucket and the paddle with a sheep on it. The 3 DOF orientation measurements of the XSens MT9 device are used to enhance the orientational part of the spatial relationship between the bucket that we assume to be world-fixed and the laptop. Collisions between the bucket and the paddle have to be detected, in addition, the viewer component on the laptop has to show the virtual sheep on the paddle and the dye in the bucket, thus inferences from the laptop to these objects have to be made, too. Figure 7.3 shows the resulting SR graph.

Global Coordinate System. In Ubiquitous SHEEP, no global coordinate system exists. Instead, the three SR graphs just described are separate and in no explicit

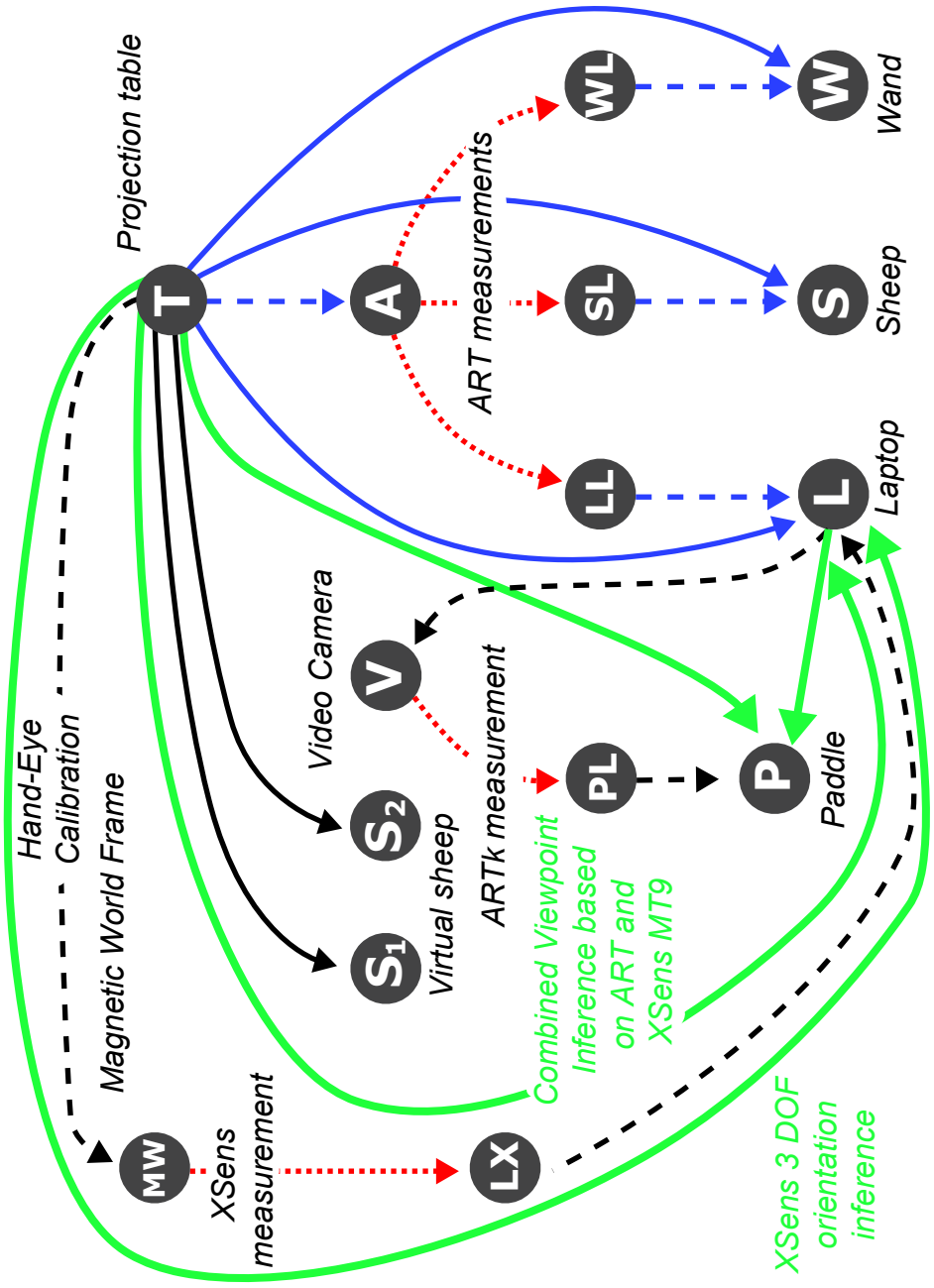


Figure 7.1: SR graph of the AR Lab setup of Ubiquitous SHEEP. The graph is similar to the original SHEEP graph depicted in figure 3.16. Note that a video camera tracking a paddle has been added, with the goal of detecting collisions between the paddle and any virtual sheep or the projection table. Real time measurements are indicated by dotted edges, necessary inferences are indicated by green edges. Static calibration relationships are indicated by dashed edges.

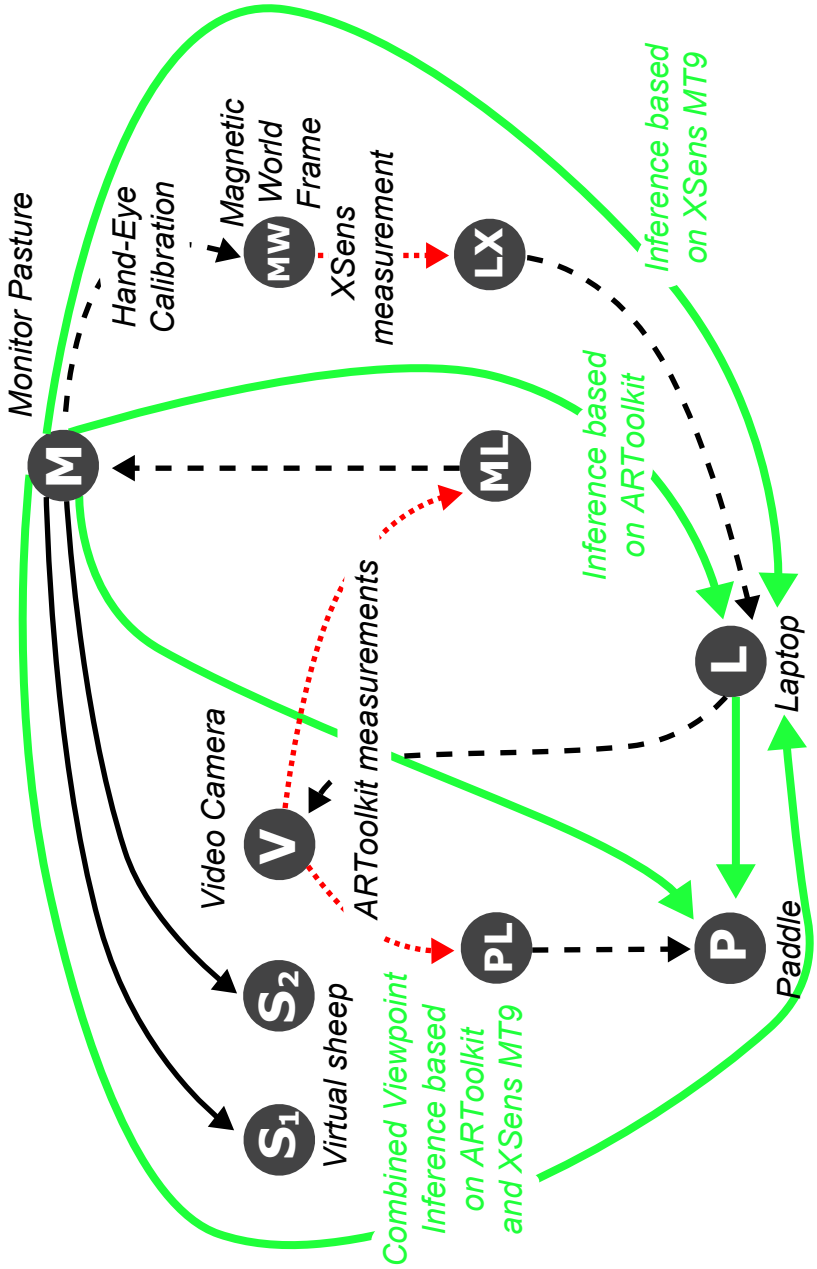


Figure 7.2: SR graph of monitor-based SHEEP application. A video camera mounted on a laptop tracks both the monitor with the pasture on it and the user's paddle. Inferences have to be made for estimating the viewpoint of the virtual camera of the laptop's scene representation and for detecting collisions between the user's paddle and the monitor pasture (green edges).

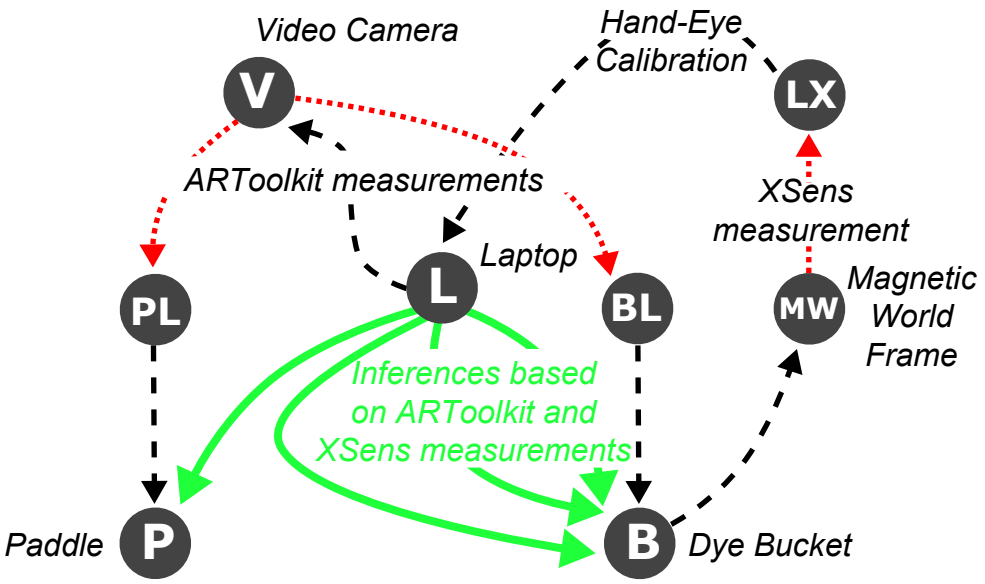


Figure 7.3: SR graph of hallway setup. The laptop needs the relative pose of the paddle and the dye bucket, in addition, collisions between these two objects have to be detected.

correlation. The scenario of Ubiquitous SHEEP does not require any correlations between them. Yet, it can be defined by the SR graph depicted in figure 7.4. This graph consists of the subgraphs depicted above, they are all put in reference to some root coordinate frame of the respective rooms. These room coordinate frames O, H and AR must in turn be put in reference to the global frame G.

Implementation Notes. Due to the current experimental nature of the DWARF Ubitrack Middleware Agent, a dynamic setup of inferences was not used. Instead, hardcoded inference components were provided.

In all locations, the mobile setup infers the spatial relationship between the laptop and the paddle:

$$L \rightarrow P = L \rightarrow V \rightarrow PL \rightarrow P$$

Additionally, the XSens MT9 measurements are combined with the results of a hand-eye calibration [33] to give a 3 DOF absolute orientation estimate of the relationship between L and some world-fixed object, depending on the user's current coarse location. This inference is combined with a 6 DOF absolute pose estimate of the same relationship.

In the AR Lab location, the existing *ObjectCalibration* service computes the inferences from the table to the objects tracked by the ART system, with the

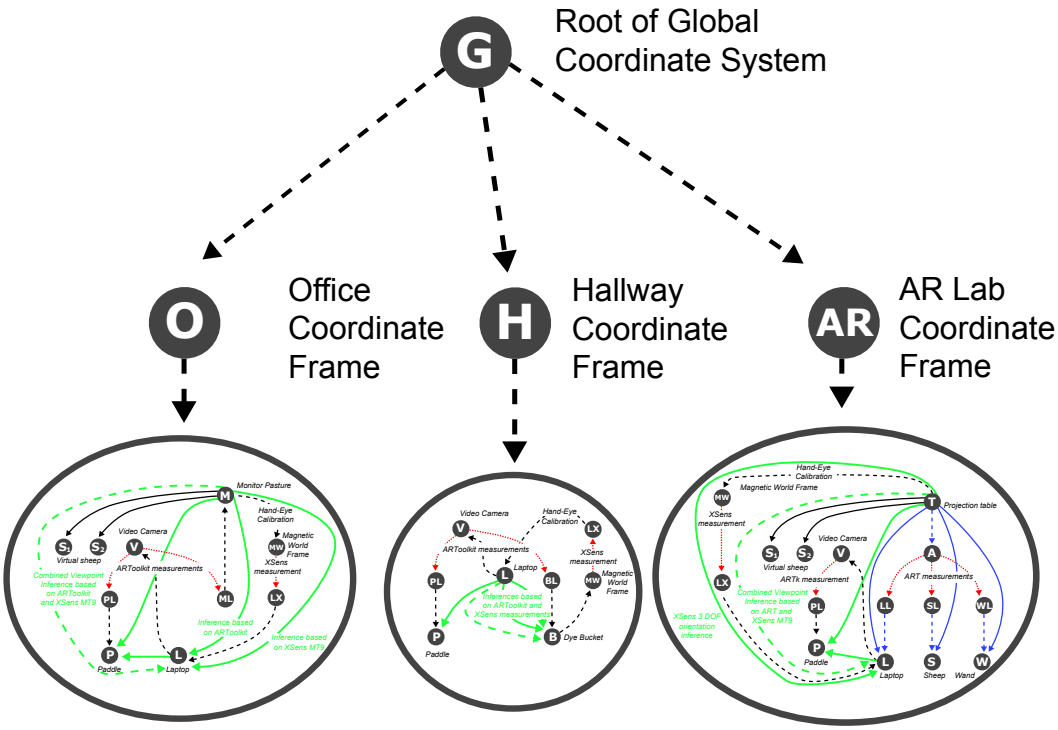


Figure 7.4: Global SR graph. Although it is not needed for Ubiquitous SHEEP, this figure shows the missing data for setting up a global coordinate system. It can be used for extending the application in new directions.

viewpoint relationship

$$T \rightarrow L = T \rightarrow A \rightarrow LL \rightarrow L$$

being the most important. An inference component reusing the two inferences above and aggregating the path

$$T \rightarrow P = T \rightarrow L \rightarrow P$$

is necessary to detect collisions between the paddle and the table and/or virtual sheep. This path is also inferred within the viewing component's scene graph for displaying a virtual sheep sitting on the paddle in the laptop augmented scenery.

In the Office location, several inferences based on ARToolkit's measurements have to be provided. The viewpoint relationship is

$$M \rightarrow L = inverse(L \rightarrow V \rightarrow ML \rightarrow M).$$

The inference $L \rightarrow P$ is reused for displaying a sheep on the paddle. Additionally, the path

$$MP = M \rightarrow L \rightarrow P$$

is necessary for detecting collisions between virtual sheep, the monitor pasture and the paddle.

In the Hallway location, in addition to the inference $L \rightarrow P$, an inference component for the relationship of the bucket to the laptop

$$L \rightarrow B = L \rightarrow V \rightarrow BL \rightarrow B$$

is created by combining measurements of the ARToolkit and the XSens tracker (dashed green edge in figure 7.3).

The integration of the XSens 3 DOF absolute orientation tracker for stabilizing the laptop's tracking results requires an inference chain for each location. In the AR Lab, the inference consists of the path

$$T \rightarrow L = T \rightarrow MW \rightarrow LX \rightarrow L$$

with MW being the magnetic world coordinate frame the XSens device makes its measurements of the laptop locatable LX in. To transform these measurements in the projection table's coordinate frame, a hand-eye calibration [33] needs to be performed and modeled as a static edge in the SR graph. In the Office location, we assume the monitor to be fixed relative to the world coordinate system and consequently have the following inference chain:

$$M \rightarrow L = M \rightarrow MW \rightarrow LX \rightarrow L$$

Finally, in the Hallway location the bucket's pose is assumed to be in a static relationship to the world, leading to the inference

$$L \rightarrow B = inverse(B \rightarrow MW \rightarrow LX \rightarrow L)$$

All orientational measurements of the XSens device are only used if the primary device (ART dTrack or ARToolkit) fails, which can be modeled by an additional inference edge that is drawn green in the figures.

Figures 7.5 and 7.6 show how the runtime data flow is organized with DWARF components. The mobile setup is constantly responsible for estimating the pose of the paddle. The environment takes care of location-specific configuration of the ARToolkit and adequate pose reconstruction. In addition, inferences based on transitive measurements and precalibrated static relationships are made.

7.3.2 Presentation Subsystem

The presentation subsystem consists of the DWARF Open Inventor based viewing component (DWARF Viewer) that operates in video see through mode on the mobile laptop, using the attached camera's video stream as background for the virtual

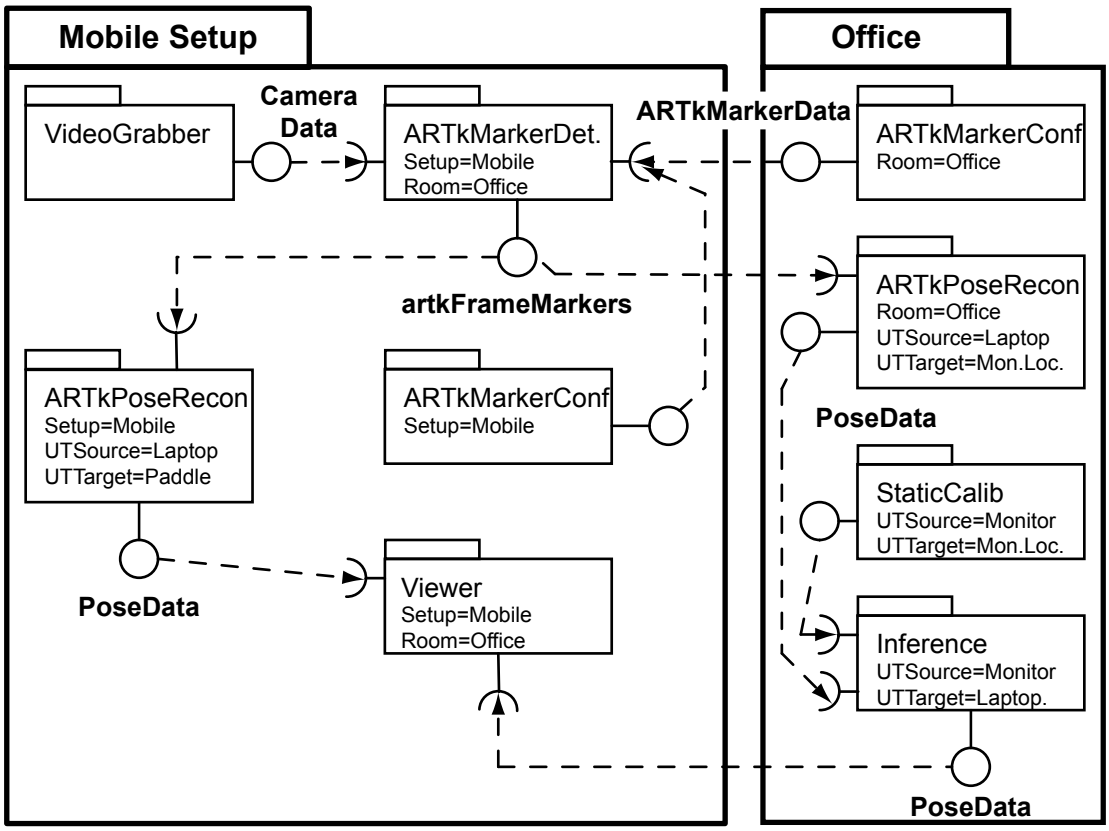


Figure 7.5: DWARF components involved in runtime spatial data flow. This UML diagram shows a representative part of the system state in location `Office`.

objects. Both in the Office and ARLab locations, the DWARF Viewer is configured in a way that displays only the virtual pasture with its sheep on top.

The DWARF Viewer offers an API to exchange or modify the current scene graph. In addition, data from the tracking subsystem can be used to dynamically update values in the scene graph's transformation nodes. There are multiple places of the default scene graph where user-defined subgraphs can be attached:

SCENEROOT: All scene graphs under the SCENEROOT hook are in fixed relationship to the scene graph main coordinate system. This hook can be used for world-fixed objects that should be drawn from a viewpoint defined by some tracking system, in our case the ART dTrack or ARToolkit systems.

CAMROOT: The pose of the CAMROOT hook is constantly kept the same as that of the virtual camera. This hook can be used to display objects that are fixed relative to the camera, in our case, the models describing the paddle and virtual sheep sitting on it are put at this hook.

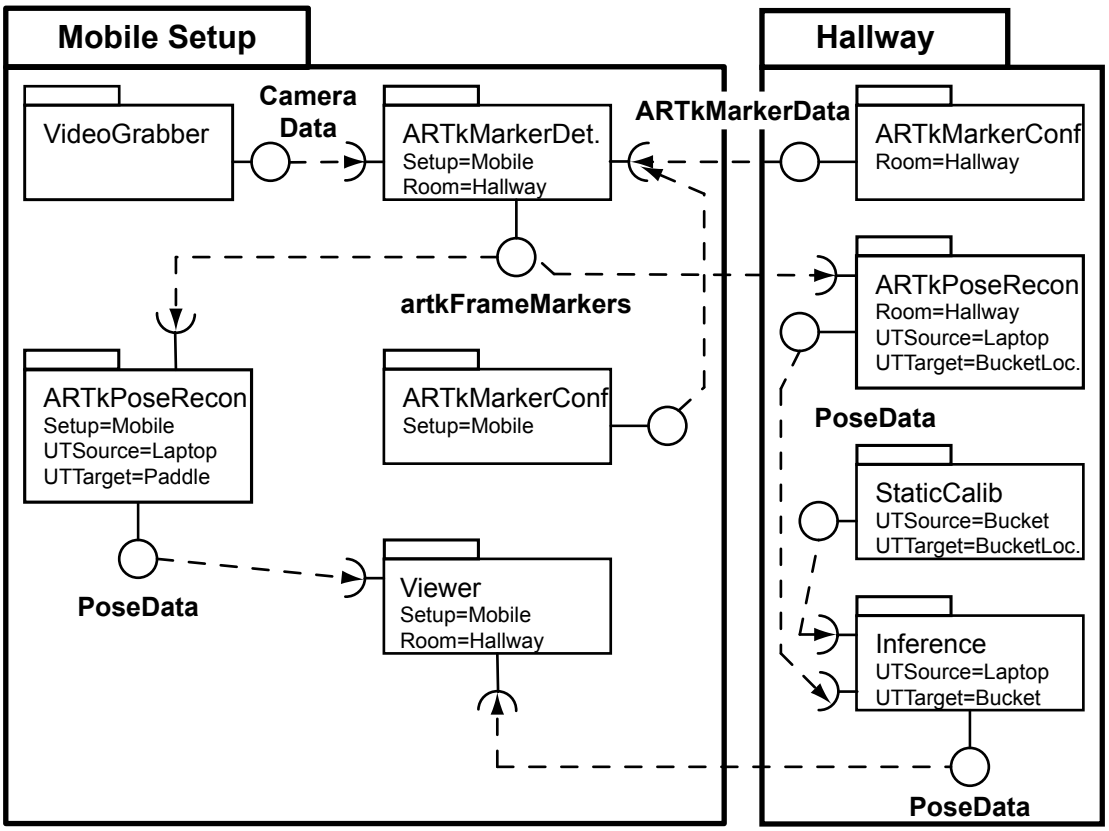


Figure 7.6: DWARF components involved in runtime spatial data flow. This UML diagram shows a representative part of the system state in location `Hallway`.

FOREGROUNDROOT: This hook offers a convenient facility to display 2D messages. Its contents are drawn using an orthogonal camera, in Ubiquitous SHEEP it is used to inform the user of tasks and the system state.

The Viewer API is used by a *ViewerController* service that configures the Viewer service according to the current environmental context.

The three contextual entities in the demo application relevant for the presentation subsystem are the same as those for the tracking subsystem. The corresponding scene graphs are shown in figures 7.7, 7.8 and 7.9.

Figures 7.10 and 7.11 show the presentation subsystem setup in the locations `ARLab` and `Hallway`, the system configuration on startup and in location `Office` can be derived analogously.

The DWARF Viewer connects to all *ViewerController* services matching its current contextual state. The controller services transmit 3D models of content to be displayed on the Viewer. Subsequently, the Viewer generates adequate needs for

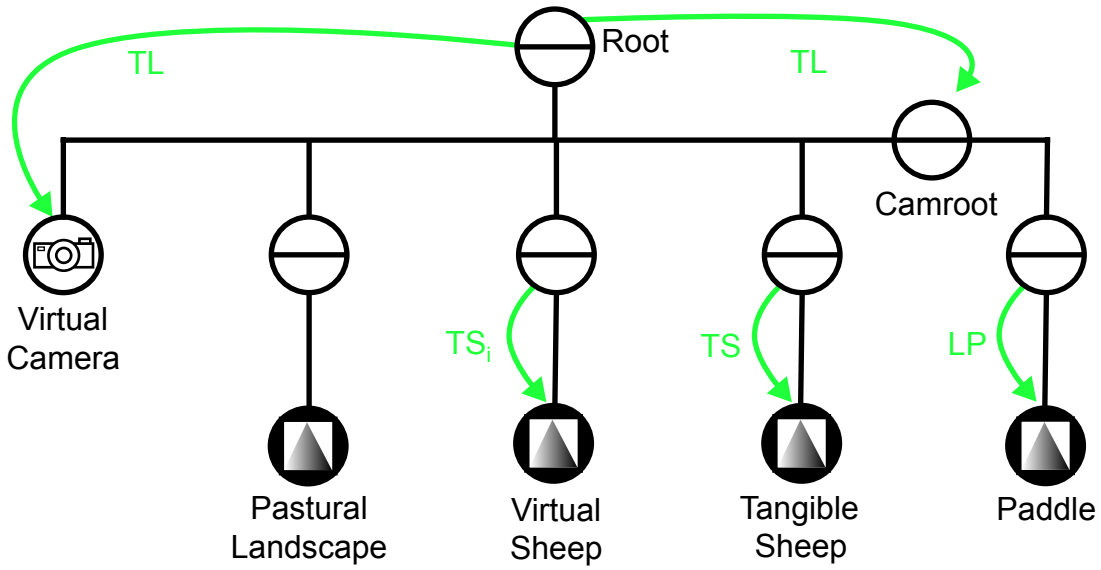


Figure 7.7: AR Lab location scene graphs shown in DWARF Viewer. The green arrows indicate which inferred data from the corresponding SR graph (figure 7.1) updates the spatial transform nodes in the scene graphs. The pastoral landscape is stationary. The virtual camera's location is set by the inference TL based on the ART dTrack system's measurements. The pose of the virtual sheep is set by the simulation relative to the world coordinate system, and the tangible sheep is set by an inference based on the ART system's measurement as well. Finally, the paddle's pose is set by a scene-graph implicit inference based on measurements from both ART and ARToolkit.

`PoseData` that are fulfilled by the tracking subsystem detailed above.

7.3.3 Interaction Subsystem

The interaction subsystem is responsible for detecting a user's request to manipulate the virtual scenery. The user has two input facilities, he can utter speech commands (as in the original SHEEP demo) and touch virtual objects with real ones. The user's only interaction device is the paddle, which he uses to pick up, color or drop sheep.

Consequently, a *collision detection* component was developed that constantly monitors the relative positions of the paddle to the projection table, the dye bucket or the monitor, depending on the user's current room context. In addition, a magic wand can be used on the projection table to kill a sheep by touching it and giving the speech command "die", or to insert a sheep by touching the projection table and

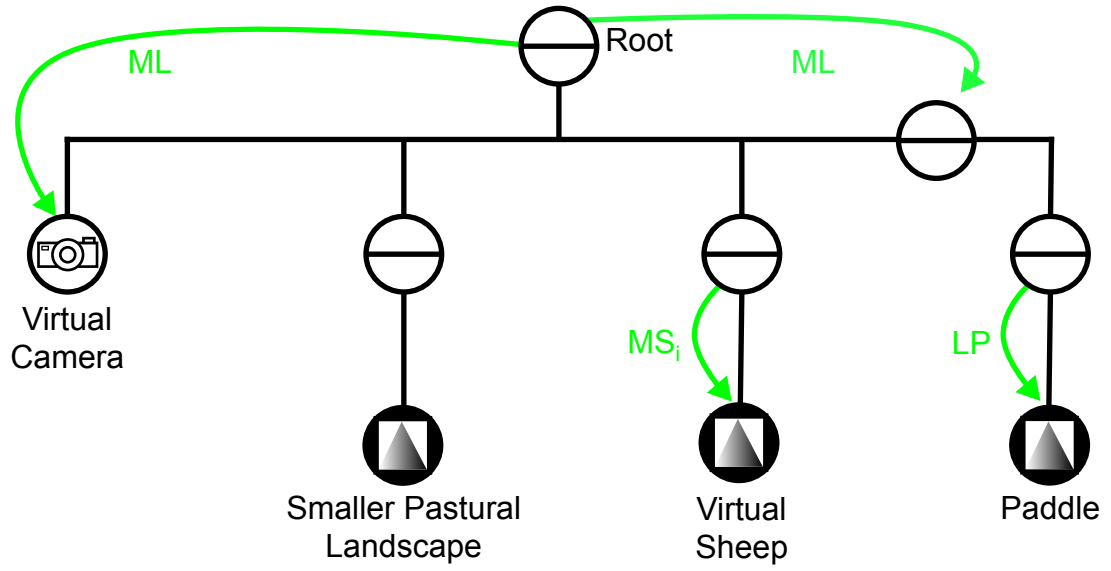


Figure 7.8: Office location scene graph shown in DWARF Viewer. Again, the pastoral landscape is stationary. The pose of the virtual sheep is set by the simulation, and both the camera's and the paddle's pose are set by inferences based on ARToolkit measurements.

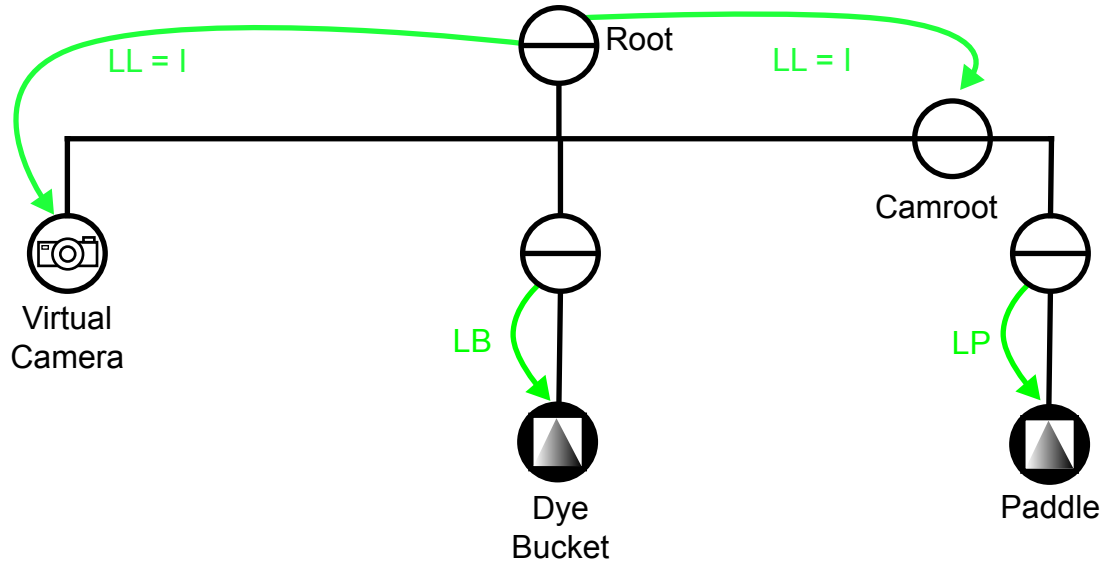


Figure 7.9: Hallway location scene graphs shown in DWARF Viewer. The camera is modeled as stationary, thus its pose relative to the root of the scene graph is identity. Both the bucket's and the paddle's poses are set based on ARToolkit measurements.

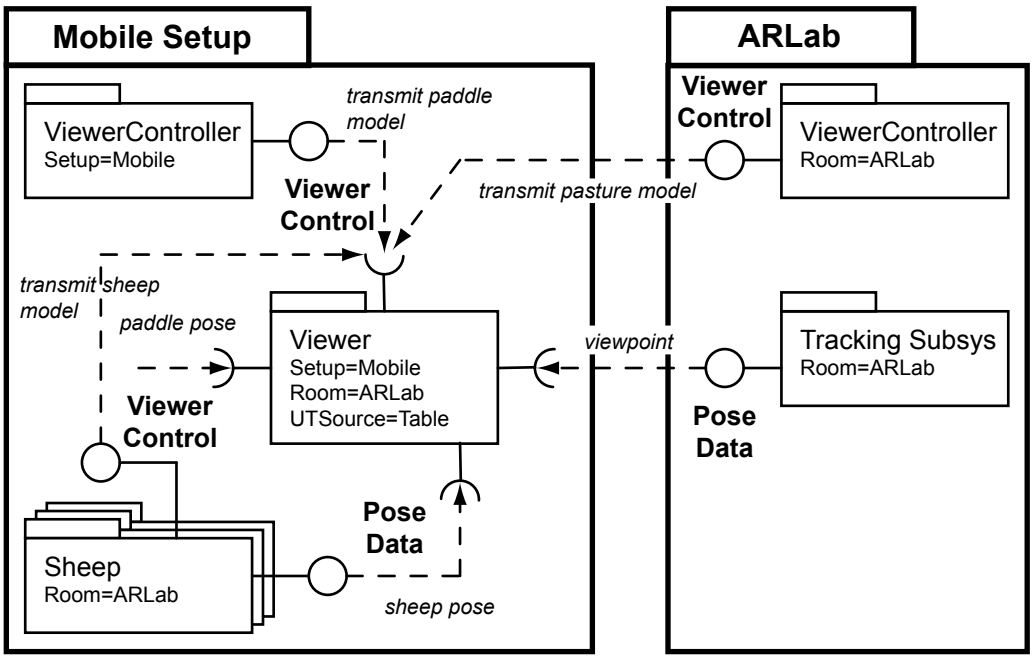


Figure 7.10: DWARF components involved in runtime reconfiguration of the mobile setup's Viewer. This UML diagram shows the system state in location `ARLab`.

uttering the command "insert". A separate, reused collision detection component is used for detecting whether the magic wand collides with the table or a virtual sheep.

To enable the multi modal combination of speech commands and collision gestures, the Petri-net based DWARF *User Interface Controller* [78] was reused.

The interactions with the paddle work in an implicit fashion: on receiving collision detection events, a *Herd Context Estimation* component switches the contextual herd state of the respective virtual sheep. As detailed in the spatial configuration architecture description of the last chapter, this leads to the disconnection from the current configuration component and reconnection with a new configuration components. These dis- and reconnections implicitly modify the virtual sheep's behavior.

7.3.4 Virtual Sheep Simulation Subsystem

The virtual sheep simulation was reimplemented, based on ideas of the original virtual sheep simulation of the SHEEP demo. The virtual sheep's task is to listen to the current herd's other sheep's poses, calculate its own movement and broadcast the resulting new pose. To make the simulation more realistic, a sheep has knowledge of its current pasture's extensions. In addition, it has knowledge about its

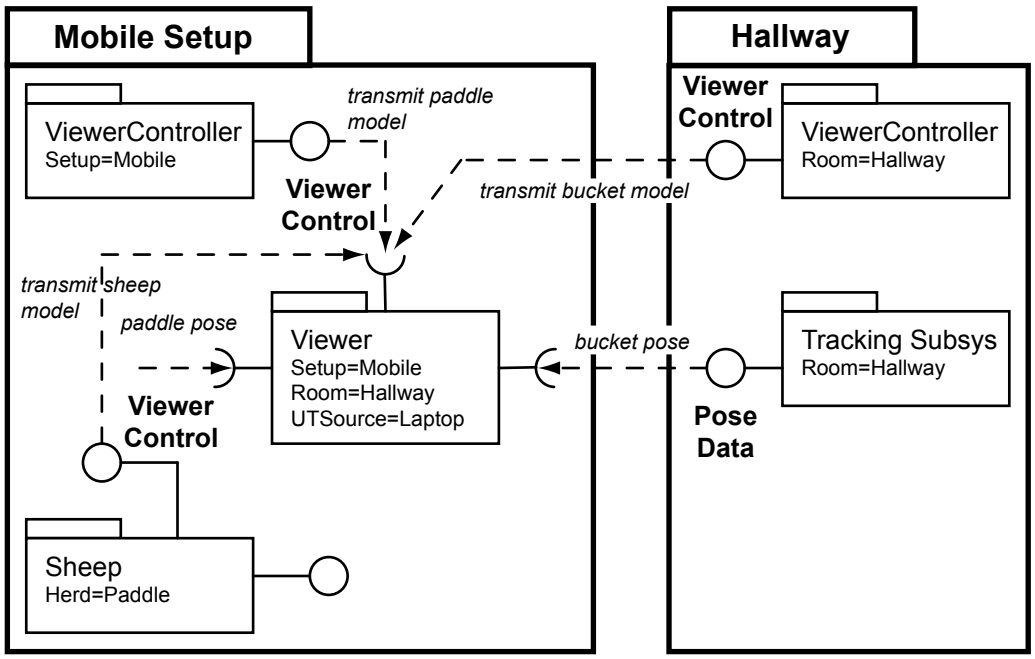


Figure 7.11: DWARF components involved in runtime reconfiguration of the mobile setup's Viewer. This UML diagram shows the system state in location `Hallway`.

appearance and broadcasts this appearance to all interested viewing components. Finally, the virtual sheep is aware of its own color and changes the color if it is told to do so by a *Sheep Coloring* service listening for collision events between the paddle and the dye bucket in the hallway. Color data changes, pasture extensions and events indicating that a sheep should die are received via a remote interface of type `SheepControl`.

Figure 7.12 shows the relevant connections if the sheep is actually on a pasture, and figure 7.13 shows the setup if the sheep is on the user's paddle.

7.3.5 Context Subsystem

The context subsystem is responsible for estimating the current relevant context and modifying the system's behavior accordingly. It employs the distributed spatial configuration architecture described in section 6.2 including the context extensions discussed in section 6.3.

The relevant context information consists of two parts. Spatially indexed information about the user's current location, i.e. the room the user is currently in, and the herd a virtual sheep is a member of. The implications of the user's room are changes to the location, interaction and tracking subsystems and have been discussed above. The consequences of a changing herd state of a virtual sheep have

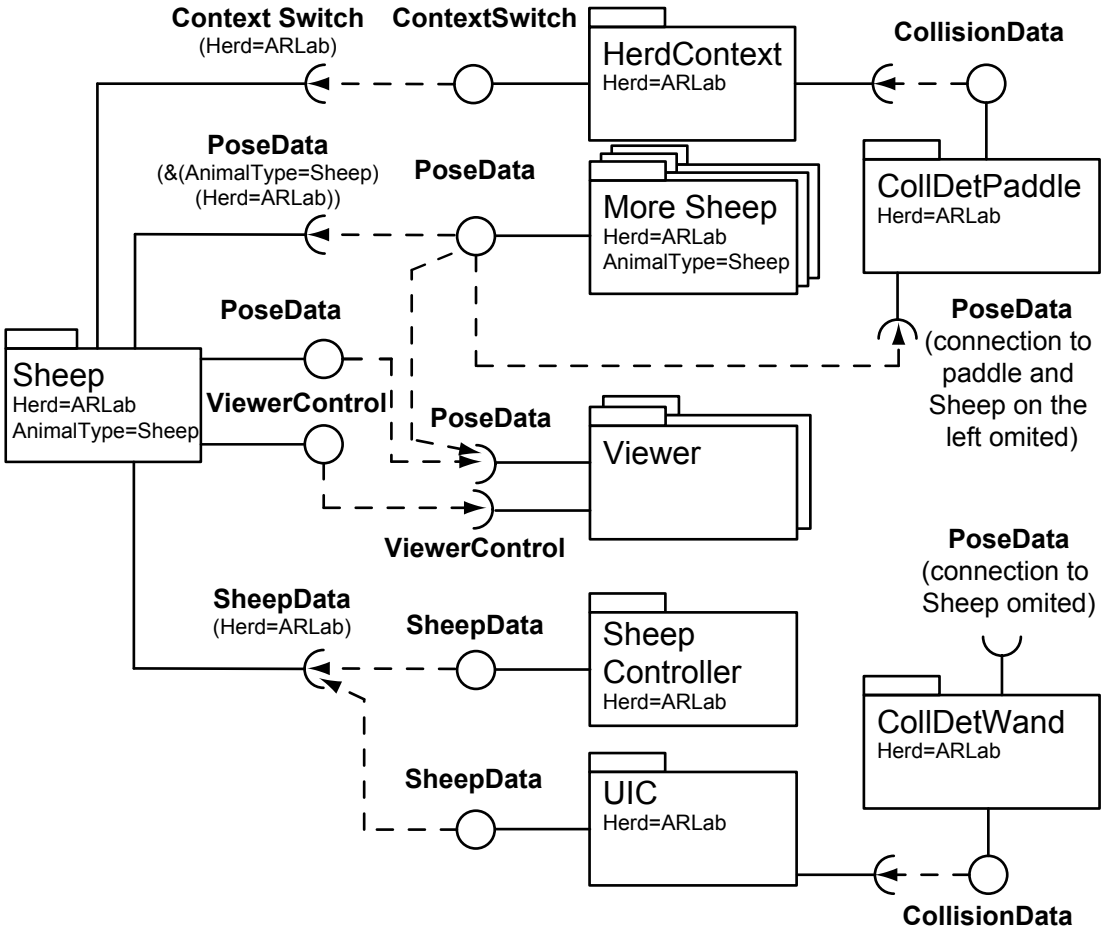


Figure 7.12: DWARF components involved in runtime virtual sheep reconfiguration if the sheep is in herd state `ARLab` or `Monitor`.

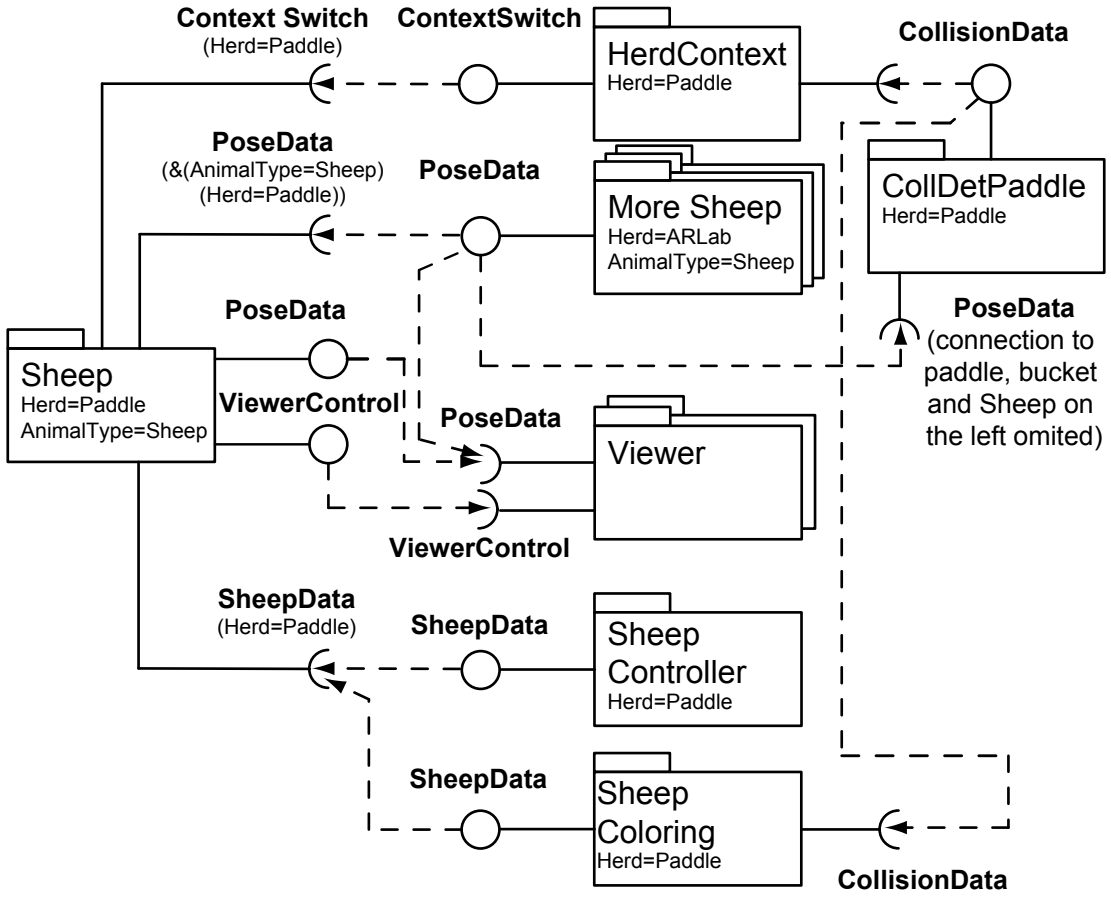


Figure 7.13: DWARF components involved in runtime virtual sheep reconfiguration if the sheep is in herd state `Paddle`.

also been described above.

Estimating the User's Room. Estimating spatially indexed context out of unobtrusive sensor data (e.g. [66]) is beyond the scope of this thesis. Consequently, we simulate the necessary contextual information via iButtons[5]. The mobile user gets a reading device and has to touch iButtons attached on both sides of all relevant doors. When the reader touches an iButton, its unique serial number is transmitted and a lookup table in the stationary environment is used to modify the context accordingly. For example, if the user touches the iButton attached to the outer side of the AR Lab's door, the context attribute `Room` is changed to `ARLab`.

Estimating a Sheep's Herd State. A change in the herd of a sheep is triggered by the interaction subsystem. The context attribute `Herd` can have three values: `ProjectionTable`, `Monitor` and `Paddle`. On startup, a virtual sheep is associated with either the herd on the monitor or the herd on the projection table. The following state changes can occur:

Projection table → paddle: This happens whenever a user picks up a sheep from the table. The transition is triggered by a collision between the paddle and a virtual sheep in herd state `ProjectionTable`.

Paddle → projection table: This state change occurs for all sheep in state `Paddle` if the paddle collides with the projection table.

Monitor → paddle: This happens whenever a user picks up a sheep from the monitor. The transition is triggered by a collision between the paddle and a virtual sheep in herd state `Monitor`.

Paddle → monitor: This state change occurs for all sheep in herd state `Paddle` if the paddle collides with the monitor.

Implementation Notes. To facilitate the implementation of context-aware services within DWARF a base class from which services should be derived was developed. It encapsulates the service's communication with the DWARF service manager. On startup of the service, it parses all ability and need descriptions. If a need of type `ContextSwitch` is found, it provides an object handling this need. The object essentially consists of an interface to set, get and delete contextual attributes, represented as string name/value pairs. If such a change occurs, the attributes of the service are changed accordingly. In addition, the predicates of all needs that have the attribute `ContextAffected` set are changed such that they only match abilities that have matching context attributes.

[5] `http://www.ibutton.com/`

The base class mechanism allows to add context-aware behavior to all dwarf services that are configurable via ability/need connections.

7.4 Results

Ubiquitous SHEEP was implemented exactly as described above, only the XSens MT9 integration is still pending. This section describes the results obtained.

Deployment. Ubiquitous SHEEP was deployed on three stationary computers, one for each room, and the mobile setup. On startup, the mobile setup only has information about the paddle's marker and consequently only can track this item. It displays a message telling the user to input the current location context by touching the next iButton (figure 7.14.

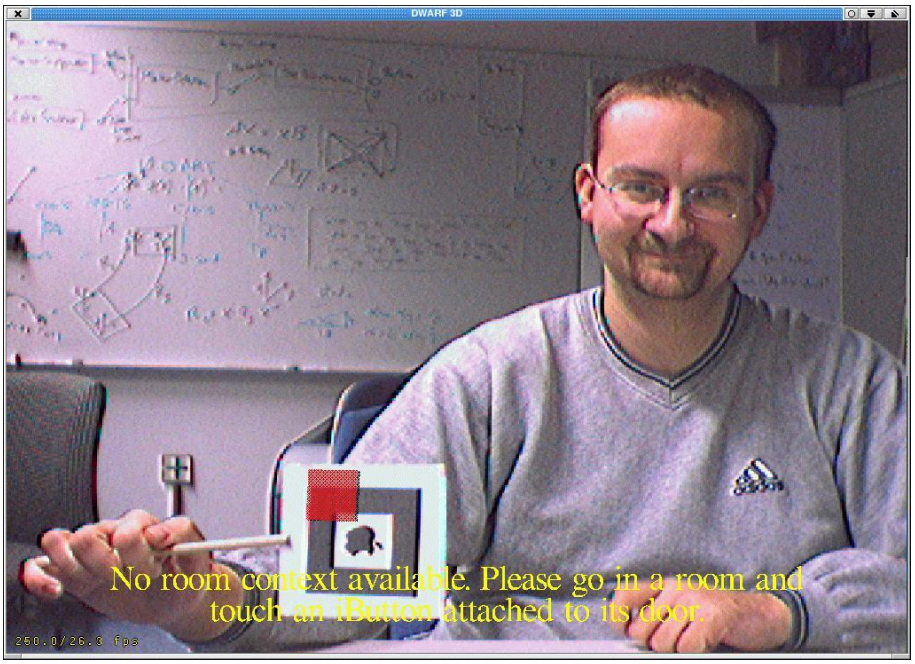


Figure 7.14: Startup scene on mobile setup. The user is told to enter the current location context via touching the next iButton.

If the user enters the AR Lab, the context of the mobile viewer is adjusted such that the projection table pasture is displayed in video-see-through-mode (figure 7.15).

In the office, the sheep are displayed on a very simple pasture on a DWARF Viewer running on a standard monitor, and the mobile setup displays a video-see-through scene of the same pasture (figure 7.16), with the viewpoint information obtained by a dynamic configuration of the ARToolkit marker detection service.

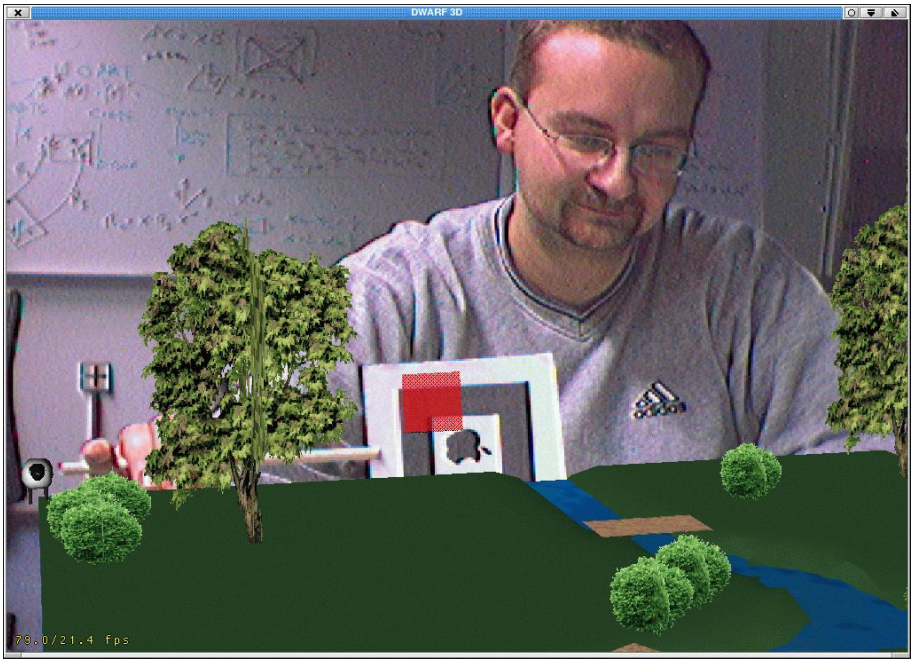


Figure 7.15: AR Lab pasture in mobile setup. The user's paddle is represented by a red semi-transparent box.

Performance. The configuration architecture proved to work successfully. As mentioned in section 6.2.5, the dependency on 3rd party service location components slows down the reconfiguration significantly. A reconfiguration on context change may take up to several seconds.

Yet, once all configuration and tracking components are connected, the resulting speed of the overall system is sufficient for the application's needs. According to MacWilliams' measurements [77], sending a single event takes less than 2 ms on the same machine, approximately 2 ms over wired Ethernet connections and 7 ms over wireless connections. Furthermore, we can assume a local shared memory connection to take less than 1 ms. As depicted in figures 7.5 and 7.6, the data flow consists of a video image transmitted via shared memory to a marker detection component (1 ms), the results of which are sent wirelessly to a pose reconstruction component (7 ms) that sends its pose data to an inference component running on the same machine (2 ms), which in turn sends its data wirelessly to the mobile Viewer (7 ms). The resulting overall additional latency of approximately 15 ms is acceptable for interactive real time AR on the laptop.

The system load of the participating computers was in a reasonable range, even the mobile setup (consisting of a Pentium 4 with 1.6 GHz and a GeForce 4MX graphics board) used only 50% of available CPU time.

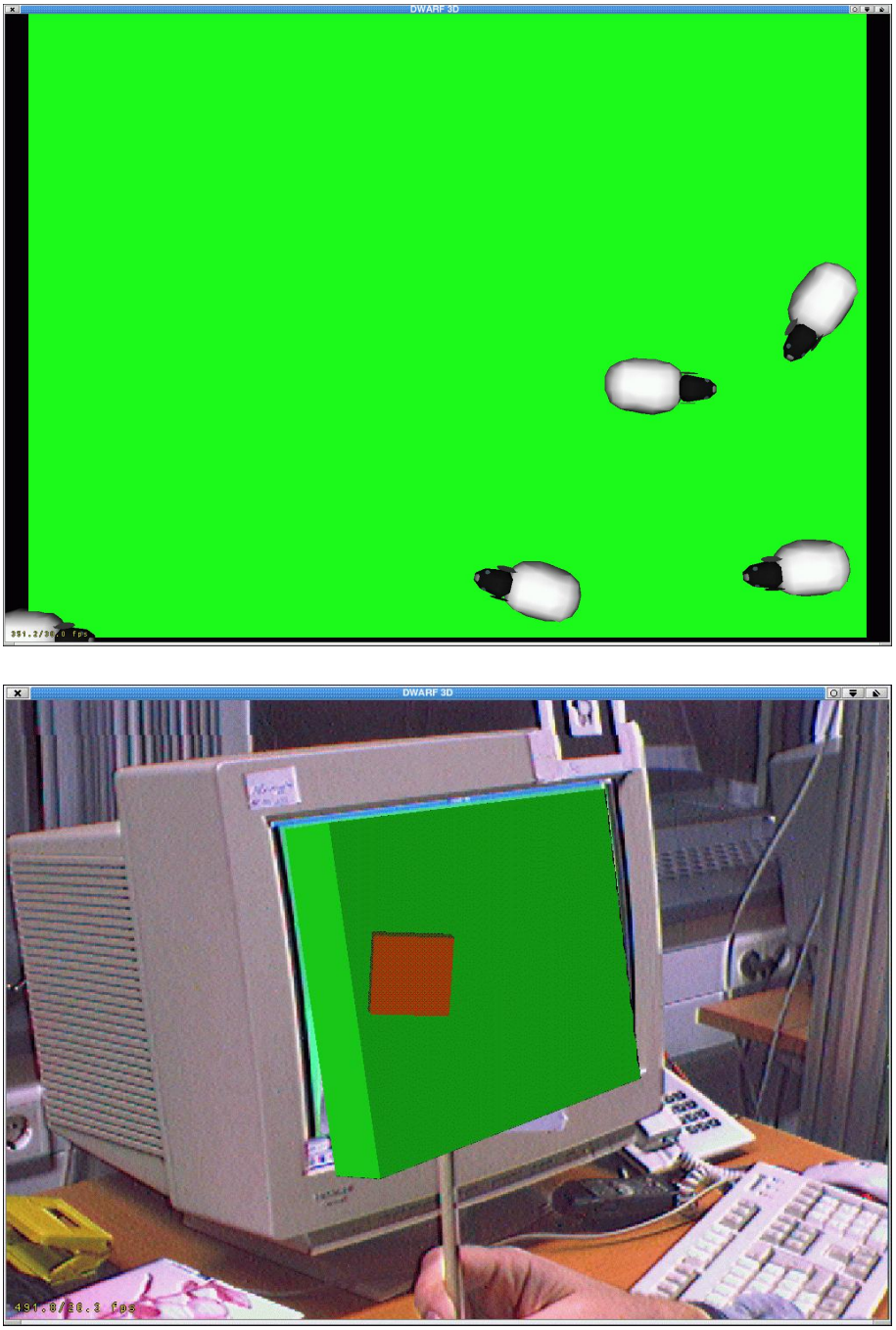


Figure 7.16: Office pasture as seen on the monitor and the mobile setup.

7.5 Discussion

The implementation of the Ubiquitous SHEEP demonstration application was a major effort, as most existing DWARF components have not been developed with the design goal of full runtime reconfigurability. Several lessons have been learned during the development process.

The Ubitrack formalism as a modeling tool: The Ubitrack formalism proved to be an excellent tool for modeling complex multi-sensor tracking setups. Changing configurations could be done in a very short time. Additionally, all necessary calibration spatial relationships can be derived immediately from the SR graphs, leading to a clear and consistent model of run time spatial data flow.

Configuration interfaces: Implementing components that are fully configurable via external context-dependent configuration components was a challenging task. The inherent multi-threaded execution of DWARF-based distributed systems required a careful handling of internal data structures. In addition, the design of configuration APIs requires a solid model of parameters influencing the transient and persistent system state.

Debugging: A massively distributed system such as Ubiquitous SHEEP which consists of more than 40 services running in parallel on multiple computers requires a careful choice of testing and debugging methodology. During the development, extensive unit tests were performed on individual components, especially the behavior of the controller components configuring the generic Sheep and Viewer services were tested thoroughly. However, much effort was necessary to debug the overall setup, as it is very hard to foresee all possible execution sequences in a multi-threaded system.

Calibration: The usability of the overall system depends heavily on the careful calibration of all spatial relationships involved. This was not the major focus of the Ubiquitous SHEEP demo, however, the tracking architecture based on the Ubitrack formalism allows to add sophisticated interactive calibration components without any change to the existing system. In fact, only the static calibration components now containing hand-tuned coarse calibration parameters must be exchanged for some interactive services.

Multi-threading issues: Ubiquitous SHEEP proves that the distributed DWARF approach to building AR systems is feasible. However, severe problems arise as soon as the sum of all components running on a computer consume 100% of the available CPU time. If this happens, it is the underlying operating system's task to schedule the scarce processing resources to the individual components of the system. Common operating systems (Linux Kernel 2.4 in

our case) are not optimized for AR systems, thus the crucial tracking data flow components may get assigned too little CPU time. If this happens, significant latencies occur, that can sum up to several seconds, essentially rendering the whole system useless.

Yet, efficient implementation and optimization of the components involved in Ubiquitous SHEEP prevented these problems.

CHAPTER 8

Conclusion

This chapter gives a summary of the work described in this thesis, draws some conclusions based on the achieved results and gives an outlook to future research opportunities.

8.1 Summary

The core contribution of the work described is threefold. In chapter 3, a formal model that allows the unified description of arbitrary sensor networks for applications in Augmented Reality and Ubiquitous Computing domains is presented. Chapters 4 and 5 describe an abstract distributed implementation concept and a real prototypical implementation based on the DWARF system. Finally, chapter 6 discusses the problem of how to integrate mobile setups in stationary tracking environments and presents a software architecture that enables an ad-hoc integration without the necessity to store knowledge about the environment on the mobile setup.

The formal model is used successfully throughout this thesis as a tool for modeling spatial relationships within tracking setups. The core algorithm of the abstract distributed implementation concept has been simulated with promising results. All architectural concepts that have been developed have been proved by working implementations. The Ubiquitous SHEEP demo described in chapter 7 shows that the concepts described in this thesis can interoperate within a single functional AR system.

8.2 Discussion

Reconsidering the problem statement and the key requirements presented in section 1.4, this thesis gives solutions to the major problems of Ubiquitous Tracking. This section discusses their limitations in scope of the general Ubitrack problem.

General Approach. The key Ubitrack problem consists of an automated abstraction of sensors for location tracking. The approach presented in this thesis consists of algorithms and architectures using a graph-based formal model.

Only a formal basis allows the algorithmic treatment of multi-sensor setups. Previous approaches either employed highly specialized knowledge to fuse well-known sensors [41, 49] without applicability to general sensors, restricted the application area to a specific set of sensors [59, 94, 99], thus facilitating the underlying formalism massively, or focused on abstract modeling of multi-sensor location tracking setups without a detailed formal model [34, 48].

Care has been taken to design the formalism in a way that allows an efficient implementation without restricting the formalism's expressiveness. The restrictions introduced by the proposed implementation concepts are discussed in the respective chapters, they can be summarized by the assumption that the quality of the participating sensors' estimates and the topology of their distribution have to change less frequently than the estimated spatial relationships by some orders of magnitude.

Formal Model. The Ubitrack formal model is hard to assess. One could think of several alternatives, and it is impossible to strictly prove the optimality of the given Ubitrack model. However, no other formalism with the same broad scope has been proposed to date, as such, the Ubitrack formalism can be evaluated according to two criteria: expressiveness and simplicity. The formalism must be expressive enough to allow modeling all problems of the Ubitrack domain and simple enough to make use of it without having to give specifications that need more time to set up than hand-crafted sensor fusion solutions.

To ensure maximum expressiveness, a directed graph is used to model spatial relationships. This allows the modeling of all kinds of objects and their relationships among each other. With the concept of attributing arbitrary functions of time to the graph's edges it is possible to describe all kinds of spatial relationships within the formalism. However, how to combine diverse spatial relationships has not been a focus of this thesis and needs further efforts.

A multitude of examples how the formalism can be used to treat real-world tracking problems was given throughout this thesis (see sections 3.3, 4.7, 6.4 and 7.3.1). Especially the conversion of the existing SHEEP demo with its many implicit assumptions on the underlying spatial relationships to the Ubiquitous SHEEP demo with an explicit representation of the spatial relationships shows that the proposed

graph-based approach leads to intuitive and flexible models of spatial relationships. Again, only further work can proove that the proposed formalism is suitable for the full Ubitrack problem domain. This thesis shows that at least common distributed AR applications can profit from the formal approach.

Distributed Implementation Concept. The concept described in chapter 4 allows to use the Ubitrack formalism in real systems. Along with the prototype DWARF-based implementation of chapter 5 it demonstrates the feasibility of using a formal model of spatial relationships for automated setup of multi-sensor location tracking setups.

The implementation concept and the DWARF-based implementation restrict the generality of the Ubitrack formalism. However, the resulting limitations in the degree of changes in sensor quality and topology do not harm most real-world AR and ubicomp setups.

Distributed Configuration Architecture. Chapter 6 discussed the problems arising when mobile setups should be dynamically integrated in stationary environments without a priori knowledge of its properties.

For this purpose, a distributed configuration architecture focused on setting up location tracking components was developed and later extended to general contextual components. This architecture allows to deploy contextual information in a spatially distributed way and thus segregates the syntactic information estimated by potentially mobile sensors from the semantic information generated by context estimators.

In contrast to previous context-aware architectures discussed in section 2.3, this thesis focusses on the low-level location aspect of contextual information. By distributing location and other contextual information spatially, the proposed architecture not only reduces the information overload of the user by enabling implicit interaction, but also keeps the information to be processed by the underlying computational infrastructure within reasonable amounts—only if the relevant parts of the current spatial relationship graph designed according to the Ubitrack formal model are relatively small is it possible to compute optimal spatial relationships in an efficient yet fully decentralized manner.

8.3 Future Work

Ubitrack opens a new problem domain that gives multiple opportunities for further research. Up to now, highly dynamic multi sensor tracking setups have not been investigated thoroughly. This thesis is a first step in exploring the problem domain, however, much more should be done to enable scenarios making use of omnipresent tracking information.

Extend the Formal Model. The Ubitrack formalism was designed to cope with a variety of measurement state spaces. Yet, this thesis mainly deals with a 6 DOF absolute pose state space, as is common with typical AR sensors. Most ubicomp sensors are widely different. Ways for accurate yet efficient modeling of diverse sensor networks have to be found, including semi-automated procedures for transforming varying measurement state spaces into each other.

As with the measurement state space, the Ubitrack formalism's evaluation function is defined in a general way but has been restricted throughout this thesis. Notably setups with a variety of measurement state spaces need evaluation functions that can cope with this situation. It is an open problem how to define an evaluation function that can both be applied to a broad range of applications and be parameterized in a way that allows an efficient implementation.

Another area of major future work consists of finding ways to incorporate advanced sensor fusion schemes. Up to now, the formal model and the implementations built on top of it allow to generically use the inherent transitivity of spatial relationships. Although defining generic filtering schemes based on more complex theoretical models such as the Kalman filter is an overly complex task, it should be possible to define certain assumptions that allow an automated setups of restricted filtering algorithms.

Distributed Implementation Concept. The distributed implementation concept described in this thesis allows a realization of the Ubitrack formalism especially for AR setups of moderate size. However, its scalability is clearly limited such that modifications are necessary to enable the implementation to cope with the massive amount of sensors to be expected within ubicomp environments.

Finding shortest paths in a SR graph takes exponential time in the number of nodes. If nodes are grouped into *supernodes* (similar to superpeers in Gnutella [91]) according to a natural location hierarchy, the search time in large networks could be reduced drastically, without sacrificing too much accuracy.

Finding nodes in the SR graph is out of scope of this thesis, as it is similar to the general service discovery problem in P2P systems. However, the inherent hierarchy in spatial information (country, town, street, building, floor, room, etc.) could be used to facilitate service and SR graph node discovery. Thus, the service location process would be linked to a supernode hierarchy.

Currently, optimal paths in the SR graph are only found according to some shortest path algorithm. This is a viable solution if only generic inferences based on the transitivity property of spatial relationships are to be made. If more complex filtering and sensor fusion schemes are to be employed automatically, multi-commodity max flow/min cost algorithms [4] may be a viable solution to the optimal path search problem in the SR graph. However, modeling Ubitrack problems as multi-commodity flow problems requires much work, both on the Ubitrack formalism and

on implementation concepts.

In the current concept, no optimality of shortest path search results is guaranteed. To provide a best effort approach, recomputations of shortest paths are triggered at constant time intervals, and if the shortest path computed for some inference chain changes, the corresponding inference components are reconfigured. It would be worthwhile to find some formal way to model recomputation triggers. For example, whenever a new sensor becomes available in the working area of an application, recomputation of all inferences of this application should be triggered. This would both enhance the quality of spatial relationship inferences and reduce the computational effort, as recomputations would only be made when necessary.

Ubitrack Implementation. The DWARF-based implementation served as a proof of concept for the applicability of both the Ubitrack formalism and the distributed implementation concept. It should be extended in several directions to allow the envisioned scalability, performance and flexibility.

In the current implementation, the process of creating inference components is not optimized. Whenever an application puts a query, a new component gets instantiated that might be reused by later inference components. As such, the efficiency and degree of reuse of all inferences made by a Ubitrack implementation depends on the sequence of application queries. It would be better to make some optimizations to reduce the overall computational time, network bandwidth or other criteria by carefully balancing the computational efforts on the involved network hosts. In general, this is a very challenging problem, but it should be possible to find some special cases where existing work (e.g. [72]) could be modified to allow optimized inferences.

On the code level, several enhancements can be realized with minor conceptual effort. Currently, every DWARF service manager is accompanied by a Ubitrack Middleware Agent (UMA), and both middleware components exchange a lot of information. Thus, it would be preferable to couple them tightly. Integrating the UMA into the DWARF service manager could speed up the setup process of new inferences significantly. At runtime, all DWARF components for tracking use CORBA notification events to exchange data. This introduces a significant overhead and latency. In contrast, existing highly efficient systems such as OpenTracker use a single process and a data flow architecture with UDP-based network communication, but do not offer the component based flexibility of the DWARF approach. It should be possible to use the DWARF service concept for modeling an abstract representation of the tracking setup and nevertheless implement the runtime setup as a data flow graph within a single process space.

Integration of Mobile Setups. The distributed spatial configuration architecture used to provide ad-hoc integration of mobile setups depends heavily on how

contextual information is structured. This is an open research problem, and new developments such as the *context cube* [46] should be investigated. Currently, all contextual information must be generated manually. This strategy is not promising for large-scale applications, the amount of information gets too big. The viability of combining automated context classification schemes with the proposed configuration architecture has to be investigated.

Building upon ideas from Lester [71], we made first experiments employing a frequency analysis of motion patterns in order to identify anonymous locatables [114]. Anonymous locatables with specific motion patterns could be particularly useful for the automated processing of natural feature tracker output.

Applications. The scenarios discussed in the introduction give a rough first idea how the concepts developed within this thesis could help real-world applications. Yet, both AR and ubicomp research is in an early stage and it is not clear which application areas will be the most promising. Thus, finding application domains where the Ubitrack concepts come in handy is a major part of future work.

With the large number of sensed data concerning the location of people, security and privacy issues will require major attention when bringing Ubitrack applications to the real world. The concept of distributed storage of data is a solid basis for letting every user decide which information gets released to the public, but much conceptional work still has to be done in this area.

Summary. This thesis has explored the new problem domain of Ubiquitous Tracking. Merging AR and ubicomp seems a promising approach to make full use of intelligent environments that will inevitably pervade people's lives within the next years.

To render the presented concepts useful, additional efforts are necessary, ranging from improvements to the formalism to code-level optimizations.

Glossary

6DOF. Six degrees of freedom. The state of a rigid object in three dimensional space containing both position and orientation. See *Pose*.

Ad-hoc Network. An *ad-hoc network* is a self-configuring network of mobile computers. The resulting network topology is arbitrary and may change rapidly and unforeseeable, as the users carrying the mobile computers are free to move randomly. The Ubitrack problem treated in this thesis tries to provide a middleware for ad-hoc sensor networks. See *Ubiquitous Tracking*.

Augmented Reality(AR). *Augmented Reality(AR)* systems mix artificial and realistic impressions such that the user is supported in performing real-world tasks. According to Azuma [8], an AR system combines real and virtual objects in a real environment, runs interactively and in real time and registers (aligns) real and virtual objects with each other in three dimensions.

Calibration. *Calibration* is the determination of the correct value of some state variable based on readings of some sensors. Within this thesis and the problem domain of location estimation, the term calibration refers to the computation of parameters that change seldom at runtime and need a special calibration procedure for computation. See *Tracking*.

Context. 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 (definition from Dey [34]).

Context Aware Computing. A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task (definition from Dey [34]).

CORBA. The *Common Object Request Broker Architecture (CORBA)* is a standard created and controlled by the Object Management Group (OMG)[1]. It defines APIs and communication protocols that allow distributed applications running on a variety of operating systems and written in a variety of object-oriented languages to communicate transparently. CORBA is the basic middleware underlying the DWARF framework and therefore the distribution architectures described throughout this thesis. See *DWARF*.

Distributed Computing. A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages (definition from Coulouris et al. [32]). The Ubitrack problem researched in this thesis is inherently distributed. See *Ubiquitous Tracking, CORBA, DWARF*.

DWARF. The *Distributed Wearable Augmented Reality Framework (DWARF)* is a research platform for distributed AR applications. It employs a set of distributed components to model AR applications. The DWARF middleware is based on CORBA and serves as the basis of the distributed architectures described in this thesis. See *CORBA*.

Global Positioning System (GPS). The Global Positioning System (GPS) is a satellite navigation system used for determining the location of an object to which a cell phone-sized receiver is attached. It works almost anywhere on Earth as long as a clear view to the sky is available and provides an accuracy of about 10m. Within this thesis, GPS receivers are treated as sensors with a 3 DOF position measurement state space.

Kalman Filter. The *Kalman filter* is a set of mathematical equations that work in a predictor-corrector scheme. It provides an estimator of an observed system's state that is optimal in the sense that the accumulated error covariance is minimized. Optimality can only be guaranteed if some very restricting assumptions hold (such as the underlying process being linear), yet, it works well in many real-world problems, especially in tracking moving objects.

Lag. See *latency*.

Latency. The time delay between an object's state in the real world and when information about this state is available to the computing system. Also known as *lag*.

Locatable. A *locatable* is an object that can be located by a sensor. The sensor yields the spatial relationship of the locatable relative to some reference coordinate system. See *Sensor* and *Tracker*.

[1] `http://www.omg.org/`

Measurement State Space. A sensor can only give estimates of parts of the full state of an observed real world entity. The *measurement state space* of a sensor characterizes the state subspace a given sensor can provide estimates of. See *Object State Space*.

Middleware. *Middleware* consists of software agents acting as an intermediary between different application components. In the scope of this thesis, two layers of middleware can be distinguished: first, the DWARF AR framework provides middleware that enables the communication between the various parts of the sensor abstraction and mobile configuration architectures described in this thesis, second, these architectures themselves act as middleware for location- and context-aware applications built on top of them.

Object State Space. An *Object State Space* is the full state space an object can have, e.g. its location, current temperature, color etc. Sensors can only provide estimates of a subspace of the full object state space. See *Measurement State Space*.

Peer-to-Peer Computing (P2P). A *Peer-to-Peer (P2P)* network of computers does not rely on any central server for communication, but instead lets all client devices communicate directly among each other. A pure P2P network does not distinguish between clients and servers, and only has the notion of equal *network nodes*. A hybrid P2P network has some centralized components that communicate in a P2P fashion among each other.

Pose. The term *pose* signifies the combination of position and orientation in three dimensional space, i.e. a six degrees of freedom state of a rigid object in three dimensions.

Sensor. A *sensor* is an active piece of hardware that detects the spatial relationship between a reference coordinate system and one or several objects. Using some software, the data delivered by the sensor is made available to a computer system making use of the spatial relationship. See *Locatable* and *Tracker*.

Tracker. A *tracker* is a *sensor* used for three dimensional position and/or orientation sensing in AR or ubicomp applications. See *Locatable* and *Sensor*.

Tracking. For the problem domain of location estimation, *tracking* is the process of determining and continuously updating an estimate of the locational state of some tracked object. The results of a tracking algorithm can be enhanced by careful calibration of fixed system parameters. See *Calibration*.

Ubiquitous Computing (ubicomp). *Ubiquitous Computing (ubicomp)* aims at enhancing the user's experience of the real world by bringing "invisible" computers

into it. In contrast to today's computer systems that force their users to adapt to them, the vision of ubicomp is to let the computing infrastructure adapt to the user such that information is provided almost unconsciously.

Ubiquitous Tracking (Ubitrack). The problem of *Ubiquitous Tracking* consists of providing an abstraction from sensors estimating the spatial state of arbitrary objects in the real world. It is the key problem treated in this thesis.

Ubitrack Middleware Agent (UMA). The DWARF *Ubitrack Middleware Agent* is the middleware component providing Ubitrack functionality to DWARF applications. See *DWARF, Ubiquitous Tracking.*

Virtual Reality (VR). *Virtual Reality (VR)* systems aim at creating seemingly realistic, but completely artificial worlds that are primarily used to visualize and manipulate virtual data, e.g. in a design process. Many techniques from VR are reused in AR. See *Augmented Reality.*

Bibliography

[1] Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58, March 2000.

[2] Mike Addlesee, Rupert Curwen, Steve Hodges, Joseph Newman, Pete Steggles, Andy Ward, and Andy Hopper. Implementing a sentient computing system. *IEEE Computer*, 34(8):50–56, 2001.

[3] Noha Adly, Pete Steggles, and Andy Harter. SPIRIT: A resource database for mobile users. In *Proceedings of ACM CHI'97 Workshop on Ubiquitous Computing*, Atlanta, Georgia, USA, March 1997.

[4] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Upper Saddle River, NJ, USA, 1993.

[5] Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 5(2):173–198, 2001.

[6] 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.

[7] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, and Blair MacIntyre. Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(6):34–47, Nov/Dec 2001.

[8] Ronald T. Azuma. A survey of augmented reality. *Presence, Special Issue on Augmented Reality*, 6(4):355–385, August 1997.

[9] Ronald T. Azuma, Bruce R. Hoff, Howard E. III Neely, Ronald Sarfaty, Michael J. Daily, Gary Bishop, Chi Vern, Greg Welch, Ulrich Neumann, Suya You, Rich Nichols, and Jim Cannon. Making augmented reality work outdoors requires hybrid tracking. In *Proceedings of the First International Workshop on Augmented Reality (IWAR 1998)*, San Francisco, CA, USA, November 1998.

[10] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proceedings of IEEE INFOCOM (2)*, pages 775–784, March 2000.

[11] Yaakov Bar-Shalom, X.-Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., 2001.

[12] Martin Bauer, Bernd Brügge, 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.

[13] Martin Bauer, Bernd Brügge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Christian Sandor, and Martin Wagner. An architecture concept for ubiqutous computing aware wearable computers. In *International Workshop on Smart Appliances and Wearable Computing (IWSAWC)*, July 2002.

[14] Martin Bauer, Otmar Hilliges, Asa MacWilliams, Christian Sandor, Martin Wagner, Joe Newman, Gerhard Reitmayr, Tamer Fahmy, Gudrun Klinker, Thomas Pintaric, and Dieter Schmalstieg. Integrating Studierstube and DWARF. In *Proceedings of the First International Workshop on Software Technology for Augmented Reality Systems (STARS 2003)*, 2003.

[15] Martin Bauer, Martin Wagner, Marcus Toennis, Gudrun Klinker, and Verena Broy. Lehrkonzept für ein Augmented Reality Praktikum. In *1. Workshop "Virtuelle und Erweiterte Realität"*, Chemnitz, Germany, September 2004.

[16] Michael Beigl, Albert Krohn, Tobias Zimmer, and Christian Decker. Typical sensors needed in ubiquitous and pervasive computing. In *Proc. of First International Workshop on Networked Sensing Systems*, Tokyo, Japan, 2004.

[17] Dagmar Beyer. Construction of decentralized data flow graphs in ubiquitous tracking environments. Master's thesis, Technische Universität München, 2004.

[18] Mark Billinghurst, Jerry Bowskill, Jason Morphett, and Mark Jessop. A wearable spatial conferencing space. In *Proceedings of IEEE and ACM International Symposium on Wearable Computers (ISWC)*, Pittsburgh, USA, October 1998.

[19] Mark Billinghurst, Suzanne Weghorst, and Thomas Furness III. Shared space: An augmented reality approach for computer supported collaborative work. *Virtual Reality*, 3:25–36, 1998.

[20] Oliver Bimber, Bernd Fröhlich, Dieter Schmalstieg, and L. Miguel Encarnação. The virtual showcase. *Computer Graphics & Applications*, 21(6):48–55, Nov/Dec 2001.

[21] Gary Bishop, Greg Welch, and B. Danette Allen. Course 11 – Tracking: Beyond 15 minutes of thought. In *SIGGRAPH 2001 Courses*, 2001. `http://www.cs.unc.edu/~tracker/ref/s2001/tracker/`.

[22] Alex Brooks, Alexei Makarenko, Tobias Kaupp, Stefan Williams, and Hugh Durrant-Whyte. Implementation of an indoor active sensor network. In *Proceedings of the 9th International Symposium on Experimental Robotics (ISER '04)*, Singapore, June 2004.

[23] Alex Brooks and Stefan Williams. Tracking people with networks of heterogeneous sensors. In *Proceedings of Australian Conference on Robotics and Automation*, Brisbane, Australia, December 2003.

[24] Richard R. Brooks and S. Sitharamar Iyengar. Real-time distributed sensor fusion for time-critical sensor readings. *Optical Engineering*, 36:767–779, March 1997.

[25] Peter J. Brown, John D. Bovey, and Xian Chen. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, 1997.

[26] Bernd Brügge and Allen H. Dutoit. *Object-Oriented Software Engineering. Conquering Complex and Changing Systems.* Prentice Hall, Upper Saddle River, NJ, 2000.

[27] Barry L. Brumitt, John Krumm, Brian Meyers, and Steven Shafer. Ubiquitous Computing and the Role of Geometry. *IEEE Personal Communications*, pages 41–43, October 2000.

[28] Barry L. Brumitt, Brian Meyers, John Krumm, Aamanda Kern, and Steven Shafer. EasyLiving: Technologies for intelligent environments. In *Proc. of 2nd Intl. Symposium on Handheld and Ubiquitous Computing*, 2000.

[29] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Department of Computer Science, Dartmouth College, Hanover, NH, USA, 2000.

[30] Youngkwan Cho and Ulrich Neumann. Multi-ring color fiducial systems for scalable fiducial tracking augmented reality. In *Proceedings of the IEEE 1998 Virtual Reality Annual International Symposium (VRAIS '98)*, 1998.

[31] Brian Clarkson. *Life Patterns: structure from wearable sensors*. PhD thesis, MIT Media Lab, 2002.

[32] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Pearson Education Ltd, Harlow, England, 3rd edition, 2001.

[33] Konstantinos Daniilidis. Hand-eye calibration using dual quaternions. *International Journal on Robotics Research*, 18:286–198, 1999.

[34] Anind K. Dey. *Providing Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, 2000.

[35] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *Workshop on the What, Who, Where and How of Context-Awareness, affiliated with CHI 2000*, The Hague, Netherlands, April 2000.

[36] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[37] Florian Echtler, Fabian Sturm, Kay Kindermann, Gudrun Klinker, Joachim Stilla, Joern Trilk, and Hesam Najafi. The intelligent welding gun: Augmented reality for experimental vehicle construction. In S.K Ong and A.Y.C Nee, editors, *Virtual and Augmented Reality Applications in Manufacturing, Chapter 17*. Springer Verlag, 2003.

[38] Hans Jacob S. Feder, John J. Leonard, and Christopher M. Smith. Adaptive mobile robot navigation and mapping. *International Journal of Robotics Research*, 8(7):650–668, July 1999.

[39] Steven Feiner, Blair MacIntyre, Tobias Hoellerer, and Anthony Webster. A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. In *Proceedings of the First International Symposium on Wearable Computers (ISWC)*, pages 74–81, Cambridge, MA, October 1997.

[40] Dieter Fox, Jeffrey Hightower, Lin Liao, Dirk Schulz, and Gaetano Borriello. Bayesian filters for location estimation. *IEEE Pervasive Computing*, 2(3):24–33, July-September 2003.

[41] Eric Foxlin, Michael Harrington, and Yury Altshuler. Miniature 6-DOF inertial system for tracking HMDs. In *Proceedings of Aerosense 98*, Orlando, FL, USA, April 1998.

[42] Arthur Gelb, editor. *Applied Optimal Estimation*. The MIT Press, 1974.

[43] Yakup Genc, S. Riedel, F. Souvannavonga, C. Akynlar, and Nassir Navab. Marker-less tracking for AR: A learning-based approach. In *Proceedings of 1st International Symposium on Mixed and Augmented Reality (ISMAR)*, Darmstadt, Germany, October 2002. IEEE CS.

[44] David Graumann, Walter Lara, Jeffrey Hightower, and Gaetano Borriello. Real-world implementation of the location stack: The universal location framework. In *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2003)*. IEEE Computer Society Press, Oct. 2003.

[45] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 1999)*, pages 59–68, Seattle, WA, USA, August 1999.

[46] Lonnie Harvel, Ling Liu, Gregory D. Abowd, Yu-Xi Lim, Chris Scheibe, and Chris Chatham. Context cube: Flexible and effective manipulation of sensed context data. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing: Second International Conference, PERVASIVE 2004*, volume 3001 of *Lecture Notes in Computer Science*, pages 51–68, Linz/Vienna, Austria, April 2004. Springer-Verlag Heidelberg.

[47] Jeffrey Hightower and Gaetano Borriello. A survey and taxonomy of location systems for ubiquitous computing. Technical Report UW-CSE 01-08-03, University of Washington, Seattle, WA, USA, August 2001.

[48] Jeffrey Hightower, Barry L. Brumitt, and Gaetano Borriello. The location stack: A layered model for location in ubiquitous computing. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002)*, pages 22–28, Callicoon, NY, June 2002.

[49] William Hoff. Fusion of data from head-mounted and fixed sensors. In *Proceedings of First IEEE International Workshop on Augmented Reality*, San Francisco, CA, USA, November 1998.

[50] 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.

[51] Richard L. Holloway. Registration error analysis for augmented reality. *Presence, Special Issue on Augmented Reality*, 6(4):413–432, August 1997.

[52] *Proceedings of 1st International Symposium on Augmented Reality (ISAR)*, Munich, Germany, October 2000. IEEE CS.

[53] *Proceedings of 2nd International Symposium on Augmented Reality (ISAR)*, New York, NY, USA, October 2001. IEEE CS.

[54] *Proceedings of 1st International Symposium on Mixed and Augmented Reality (ISMAR)*, Darmstadt, Germany, October 2002. IEEE CS.

[55] *Proceedings of 2nd International Symposium on Mixed and Augmented Reality (ISMAR)*, Tokyo, Japan, October 2003. IEEE CS.

[56] *Proceedings of 3rd International Symposium on Mixed and Augmented Reality (ISMAR)*, Arlington, VA, USA, November 2004. IEEE CS.

[57] Xiaodong Jiang, Nicholas Chen, Jason Hong, Kevin Wang, Leila Takayama, and James Landay. Siren: Context-aware computing for firefighting. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing: Second International Conference, PERVASIVE 2004*, volume 3001 of *Lecture Notes in Computer Science*, pages 87–105, Linz/Vienna, Austria, April 2004. Springer-Verlag Heidelberg.

[58] Simon Julier, Yohan Baillot, Marco Lanzagorta, Dennis Brown, and Lawrence Rosenblum. BARS: Battlefield augmented reality system. In *NATO Symposium on Information Processing Techniques for Military Systems*, Istanbul, Turkey, October 2000.

[59] Michael Kalkusch, Thomas Lidy, Michael Knapp, Gerhard Reitmayr, Hannes Kaufmann, and Dieter Schmalstieg. Structured visual markers for indoor pathfinding. In *Proceedings of the First IEEE International Workshop on ARToolKit (ART02)*, Darmstadt, Germany, October 2002. IEEE Computer Society.

[60] Hirokazu Kato and Mark Billinghurst. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proc. of International Workshop on Augmented Reality IWAR'99*, pages 85–94, San Francisco, CA, USA, October 1999. IEEE CS.

[61] Alonzo Kelly. Mobile robot localization from large scale appearance mosaics. Technical Report CMU-RI-TR-00-21, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, December 2000.

[62] Nicky Kern and Bernt Schiele. Multi-sensor activity context detection for wearable computing. In *European Symposium on Ambient Intelligence*, Eindhoven, The Netherlands, November 2003.

[63] Georg Klein and Tom Drummond. Robust visual tracking for non-instrumented augmented reality. In *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 113 – 122. IEEE Computer Society, October 7 – 10 2003.

[64] Gudrun Klinker, Oliver Creighton, Allen Dutoit, Rafael Kobylinski, Christoph Vilsmeier, and Bernd Bruegge. Augmented maintenance of powerplants: A prototyping case study of a mobile AR system. In *IEEE and ACM International Symposium on Augmented Reality (ISAR)*, October 2001.

[65] Dieter Koller, Gudrun Klinker, Eric Rose, David Breen, Ross Whitaker, and Mihran Tuceryan. Real-time vision-based camera tracking for augmented reality applications. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-97)*, Lausanne, Switzerland, September 1997. ACM Press.

[66] Andreas Krause, Daniel P. Siewiorek, Asim Smailagic, and Jonny Farringdon. Unsupervised, dynamic identification of physiological and activity context in wearable computing. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, White Plains, NY, USA, October 2003. IEEE Computer Society.

[67] Jack B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, 2002.

[68] Cody Kwok, Dieter Fox, and Marina Meila. Adaptive real-time particle filters for robot localization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[69] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[70] Florian Ledermann, Gerhard Reitmayr, and Dieter Schmalstieg. Dynamically shared optical tracking. In *Proceedings of the First Augmented Reality Toolkit Workshop*, 2002.

[71] 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*, volume 3001 of *Lecture Notes in Computer Science*, pages 31–50, Linz/Vienna, Austria, April 2004. Springer-Verlag Heidelberg.

[72] Marco Lohse, Michael Repplinger, and Philipp Slusallek. Dynamic distributed multimedia: Seamless sharing and reconfiguration of multimedia flow graphs. In *Proceedings of the 2nd International Conference on Mobile and Ubiquitous Multimedia (MUM 2003)*, pages 89–95. ACM Press, 2003.

[73] Paul Lukowicz, Jamie A. Ward, Holger Junker, Mathias Stäger, Gerhard Tröster, Amin Atrash, and Thad Starner. Recognizing workshop activity using body worn microphones and accelerometers. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing: Second International Conference, PERVASIVE 2004*, volume 3001 of *Lecture Notes in Computer Science*, pages 18–32, Linz/Vienna, Austria, April 2004. Springer-Verlag Heidelberg.

[74] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc, 1996.

[75] 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, Oct. 2003.

[76] Blair MacIntyre, Enylton Machado Coelho, and Simon Julier. Estimating and adapting to registration errors in augmented reality systems. In *IEEE Virtual Reality Conference 2002 (VR 2002)*, Orlando, Florida, March 2002.

[77] Asa MacWilliams. *A Decentralized Adaptive Architecture for Ubiquitous Augmented Reality Systems*. PhD thesis, Technische Universität München, 2005. To appear.

[78] Asa MacWilliams, Christian Sandor, Martin Wagner, Martin Bauer, Gudrun Klinker, and Bernd Brügge. Herding sheep: Live system development for distributed augmented reality. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, Tokyo, Japan, October 2003.

[79] Friedemann Mattern. Pervasive/ubiquitous computing. *Informatik-Spektrum*, 24(3):145–147, June 2001.

[80] Yan Meng and Hanqi Zhuang. Self-calibration of camera-equipped robot manipulators. *International Journal of Robotics Research*, 20(11):909–921, November 2001.

[81] Mathias Möhring, Christian Lessig, and Oliver Bimber. Video see-through AR on consumer cell phones. In *Proc. of International Symposium on Mixed and Augmented Reality*, Arlington, VA, USA, November 2004.

[82] Arthur G.O. Mutambara and Hugh F. Durrant-Whyte. Fully decentralized estimation and control for a modular wheeled mobile robot. *International Journal of Robotics Research*, 19(6):582–596, June 2000.

[83] Jussi Myllymaki and James Kaufman. High-performance spatial indexing for location-based services. In *Proceedings of the twelfth international conference on World Wide Web*, pages 112–117. ACM Press, 2003.

[84] Hesam Najafi, Nassir Navab, and Gudrun Klinker. Automated initialization for marker-less tracking: A sensor fusion approach. In *Proc. of International Symposium on Mixed and Augmented Reality*, Arlington, VA, USA, November 2004.

[85] Nassir Navab. Developing killer apps for industrial augmented reality. *IEEE Computer Graphics and Applications*, 24(3):16–20, May/June 2004.

[86] Joseph Newman, David Ingram, and Andy Hopper. Augmented reality in a wide area sentient environment. In *Proc. of IEEE and ACM Int. Symp. on Augmented Reality (ISAR 2001)*, pages 77–86, New York, NY, Oct. 2001.

[87] Joseph Newman, Martin Wagner, Martin Bauer, Asa MacWilliams, Thomas Pintaric, Dagmar Beyer, Daniel Pustka, Franz Strasser, Dieter Schmalstieg, and Gudrun Klinker. Ubiquitous tracking for augmented reality. In *Proc. of International Symposium on Mixed and Augmented Reality*, Arlington, VA, USA, November 2004.

[88] Joseph Newman, Martin Wagner, Thomas Pintaric, Asa MacWilliams, Martin Bauer, Gudrun Klinker, and Dieter Schmalstieg. Fundamentals of ubiquitous tracking for augmented reality. Technical Report TR-188-2-2003-34, Vienna University of Technology, 2003.

[89] Daniela Nicklas, Matthias Großmann, Thomas Schwarz, and Steffen Volz. Architecture and data model of Nexus. *GIS Geo-Informations-Systeme*, 9, 2001.

[90] Robert J. Orr and Gregory D. Abowd. The smart floor: A mechanism for natural user identification and tracking. In *Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI 2000)*, The Hague, Netherlands, April 2000.

[91] Serguei Ososkine. Search optimization in the distributed networks. Webpage `http://www.grouter.net/gnutella/search.htm`, October 2002.

[92] Wayne Piekarski, Bernard Gunther, and Bruce H. Thomas. Integrating virtual and augmented realities in an outdoor application. In *Proceedings of 2nd International Workshop on Augmented Reality (IWAR)*, San Francsico, CA, USA, October 1999.

[93] Wayne Piekarski and Bruce Thomas. ARQuake: The outdoor augmented reality gaming system. *ACM Communications*, 45(1):36–38, January 2002.

[94] Wayne Piekarski and Bruce H. Thomas. Tinmith-evo5 a software architecture for supporting research into outdoor augmented reality environments. Technical report, Wearable Computer Laboratory, University of South Australia, December 2001.

[95] Wayne Piekarski and Bruce H. Thomas. An object-oriented software architecture for 3D mixed reality applications. In *Proceedings of 2nd International Symposium on Mixed and Augmented Reality (ISMAR)*, Tokyo, Japan, October 2003. IEEE CS.

[96] Daniel Pustka. Handling error in ubiquitous tracking setups. Master's thesis, Technische Universität München, 2004.

[97] Kasim Rehman. 101 ubiquitous computing applications survey. Webpage `http://www-lce.eng.cam.ac.uk/~kr241/html/101_ubicomp.html`.

[98] Thomas Reicher. *A Framework for Dynamically Adaptable Augmented Reality Systems*. PhD thesis, Technische Universität München, 2004.

[99] Gerhard Reitmayr and Dieter Schmalstieg. An open software architecture for virtual reality interaction. In *Proceedings of the ACM Symposium on Virtual Reality Software & Technology (VRST)*, Banff, Alberta, Canada, 2001.

[100] Gerhard Reitmayr and Dieter Schmalstieg. Collaborative augmented reality for outdoor navigation and information browsing. In *Proc. of Symposium Location Based Services and TeleCartography*, 2004.

[101] Jun Rekimoto. The magnifying glass approach to augmented reality systems. In *Proceedings of Conference on Virtual Reality Software and Technology (VRST)*, Makuhari, Chiba, Japan, November 1995.

[102] Jun Rekimoto. Navicam: A magnifying glass approach to augmented reality systems. *Presence: Teleoperators and Virtual Environments*, 6(4):399–412, August 1997.

[103] Ioannis M. Rekleitis. A particle filter tutorial for mobile robot localization. Technical Report TR-CIM-04-02, McGill University, Montreal, Canada, 2003.

[104] Jannick P. Rolland, Larry Davis, and Yohan Baillot. A survey of tracking technology for virtual environments. In W. Barfield and T. Caudell, editors, *Fundamentals of Wearable Computers and Augmented Reality*, pages 67–112. Lawrence Erlbaum, Mahwah, NJ, USA, 2001.

[105] Bill N. Schilit, Norman I. Adams, and Roy Want. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, USA, December 1994. IEEE Computer Society.

[106] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavári, L. Miguel Encarnação, M. Gervautz, and Werner Purgathofer. The studierstube augmented reality project. *PRESENCE - Teleoperators and Virtual Environments*, 11(1):32–45, 2002.

[107] Albrecht Schmidt. Implicit human computer interaction through context. *Personal Technologies*, 4(2 & 3):191–199, June 2001.

[108] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. *Computers & Graphics Journal*, 23(6):893–902, December 1999.

[109] Albrecht Schmidt and Kristof Van Laerhoven. How to build smart appliances? *IEEE Personal Communications*, 8(4):66–71, August 2001.

[110] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P2001)*, Linköping, Sweden, August 2001.

[111] Gilles Simon and Marie-Odile Berger. Reconstructing while registering: A novel approach for markerless augmented reality. In *Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR)*, Darmstadt, Germany, September 2002.

[112] Gilles Simon, Andrew W. Fitzgibbon, and Andrew Zisserman. Markerless tracking using planar structures in the scene. In *Proceedings of IEEE and ACM International Symposium on Augmented Reality (ISAR)*, pages 120–128, Munich, Germany, October 2000.

[113] Andrei State, Gentaro Hirota, David T. Chen, William F. Garrett, and Mark A. Livingston. Superior augmented reality registration by integrating

landmark tracking and magnetic tracking. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 429–438. ACM Press, 1996.

[114] Franz Strasser. Bootstrapping of sensor networks in ubiquitous tracking environments. Master's thesis, Technische Universität München, 2004.

[115] Ivan E. Sutherland. The ultimate display. In *Proceedings of the IFIPS Congress*, volume 2, pages 506–508, 1965.

[116] Ivan E. Sutherland. A head-mounted three dimensional display. In *AFIPS Conference Proceedings, Fall Joint Conference*, volume 1, pages 757–764, Washinton (DC), USA, 1968.

[117] Zsolt Szalavári, Dieter Schmalstieg, Anton Fuhrmann, and Michael Gervautz. "Studierstube" – An environment for collaboration in augmented reality. *Journal of the Virtual Reality Society – "Virtual Reality : Research, Development and Application"*, 1997.

[118] 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 & Technology (VRST)*, Banff, Alberta, Canada, 2001.

[119] B. Thomas, B. Close, J. Donoghue, J. Squires, P. DeBondi, M. Morris, and W. Piekarski. Arquake: An outdoor/indoor augmented reality first person application. In *Proc. International Symposium on Wearable Computers (ISWC)*, pages 139–146. IEEE, 2000.

[120] Marcus Tönnis. Data management for augmented reality applications. Master's thesis, Technische Universität München, 2003.

[121] Jesper L. Träffs. An experimental comparison of two distributed single-source shortest path algorithms. *Parallel Computing*, 21(9):1505–1532, September 1995.

[122] Mihran Tuceryan, Yakup Genc, and Nassir Navab. Single-point active alignment method (SPAAM) for optical see-through HMD calibration for augmented reality. *Presence: Teleoper. Virtual Environ.*, 11(3):259–276, 2002.

[123] Mihran Tuceryan, Douglas S. Greer, Ross T. Whitaker, David E. Breen, Chris Crampton, Eric Rose, and Klaus H. Ahlers. Calibration requirements and procedures for a monitor-based augmented reality system. *IEEE Transactions on Visualization and Computer Graphics*, 1:255–273, September 1995.

[124] Martin Wagner. Building wide-area applications with the AR Toolkit. In *The First IEEE International Augmented Reality Toolkit Workshop*, 2002.

[125] Martin Wagner. Distributed tracking with multiple sensors for augmented reality. In *1. Workshop "Virtuelle und Erweiterte Realität" der GI-Fachgruppe AR/VR*, Chemnitz, Germany, September 2004.

[126] 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.

[127] Martin Wagner and Felix Loew. Configuration strategies of an AR Toolkit-based wide area tracker. In *The Second IEEE International Augmented Reality Toolkit Workshop*, Tokyo, Japan, 2003.

[128] Martin Wagner, Asa MacWilliams, Martin Bauer, Gudrun Klinker, Joseph Newman, Thomas Pintaric, and Dieter Schmalstieg. Fundamentals of ubiquitous tracking. In *Advances in Pervasive Computing*, pages 285–290. Austrian Computer Society, April 2004.

[129] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992.

[130] Mark Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–100, September 1991.

[131] Mark Weiser. Hot topics: Ubiquitous computing. *IEEE Computer*, 26(10):71–72, October 1993.

[132] Greg Welch and Gary Bishop. SCAAT: Incremental tracking with incomplete information. *Computer Graphics*, 31(Annual Conference Series):333–344, 1997.

[133] Greg Welch and Gary Bishop. Course 8 – An introduction to the Kalman filter. In *SIGGRAPH 2001 Courses*, 2001. `http://www.cs.unc.edu/~tracker/ref/s2001/kalman/index.html`.

[134] Greg Welch, Gary Bishop, Leandra Vicci, Stephen Brumback, Kurtis Keller, and D'nardo Colucci. The hiball tracker: High-performance wide-area tracking for virtual and augmented environments. In *Proceedings of Symposium on Virtual Reality Software and Technology*, London, UK, December 1999.

[135] Greg Welch, Gary Bishop, Leandra Vicci, Stephen Brumback, Kurtis Keller, and D'nardo Colucci. High-performance wide-area optical tracking: The hiball tracking system. *Presence: Teleoperators and Virtual Environments*, 10(1):1–21, February 2001.

[136] Suya You and Ulrich Neumann. Fusion of vision and gyro tracking for robust augmented reality registration. In *Proceedings of IEEE Virtual Reality*, pages 71–78, Yokohama, Japan, March 2001.

[137] 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.

[138] Suya You, Ulrich Neumann, and Ronald Azuma. Orientation tracking for outdoor augmented reality registration. *IEEE Computer Graphics and Applications*, 19(6):36–42, Nov/Dec 1999.