Ubiquitous Tracking for Augmented Reality

Joseph Newman[1], Martin Wagner[2]
Martin Bauer[2], Asa MacWilliams[2], Thomas Pintaric[1], Dagmar Beyer[2],
Daniel Pustka[2], Franz Strasser[2], Dieter Schmalstieg[1] and Gudrun Klinker[2]

[1] *Technische Universität Wien*
Favoritenstraße 9-11/188/2, 1040 Vienna, Austria

[2] *Technische Universität München, Fakultät für Informatik*
Boltzmannstraße 3, 85748 Garching bei München, Germany

newman@ims.tuwien.ac.at, martin.wagner@in.tum.de

Abstract

Augmented Reality (AR) provides a natural interface to the "calm" pervasive technology anticipated in large-scale Ubiquitous Computing environments. However, the range of classic AR applications has been limited by the scope, range and cost of sensors used for tracking. Hybrid tracking approaches can go some way to extending this range. We propose an approach, called Ubiquitous Tracking, in which data from widespread and diverse heterogeneous tracking sensors is automatically and dynamically fused, and then transparently provided to applications. A formal model represents spatial relationships between objects as a graph attributed with quality-of-service parameters. This paper presents a software implementation, in which a dynamic data flow network of distributed software components is thereby constructed in response to queries and optimisation criteria specified by applications. This implementation is demonstrated using a small laboratory example, and larger setups modelled in a simulation environment.

1 Introduction and Related Work

Augmented Reality (AR) has the potential to provide a natural interface to the "calm" pervasive technology anticipated in large-scale Ubiquitous Computing [30] environments. However, most AR applications have hitherto been constrained, by the working volumes of tracking technologies, to static spaces of a few cubic metres, such as the Boeing wire assembly [5] example. Systems aiming at mobility like the Touring machine [7], Sentient AR [20], or Tinmith [22] have relied on wide-area trackers, such as GPS, which provide modest levels of accuracy at low update rates. Furthermore, they assume that sensors are deployed homogeneously throughout the area of interest, resulting in tedious off-line calibration.

Significant work on sensor fusion has been done, in order to improve tracking performance and extend operational range, with pioneering work by Azuma [2] amongst others. Höllerer, Hallaway et al. [9, 12] have shown the most comprehensive approach to integrating arbitrary sensors. However, heretofore no attempt has been made to automate this integration process for distributed sensor networks.

Existing tracking middleware cannot dynamically adapt to changes in sensor infrastructure. For example, VRPN [25] implements a static network-transparent abstraction between applications and pre-defined trackers and OpenTracker [23] implements a static "pipes & filters" dataflow model for streams of sensor readings. In contrast, the initial Ubitrack software implementation is based on middleware that allows mobile users to introduce their own equipment into an AR-capable Ubicomp environment at run-time.

We propose an approach, in which diverse and widespread heterogeneous tracking sensors are automatically discovered. The goal is then to obtain an optimal estimate of arbitrary geometric relationships and their accuracy at any time, in response to an application's query of environmental state for a given definition of optimality. This estimate results from the spontaneous dynamic fusion of appropriate sensors. We call this approach *Ubiquitous Tracking* or *Ubitrack*. Such a tracker abstraction can help applications handle the varying levels of tracker uncertainty that affects the registration of virtual objects when perceived through a head-mounted display [4, 19].

Example Scenario A small example has been built to illustrate the Ubitrack concepts and provide a suitable scenario for testing implementations. A stationary tracker combined with a mobile camera, delivering images to an optical tracker, has an increased tracking range beyond that

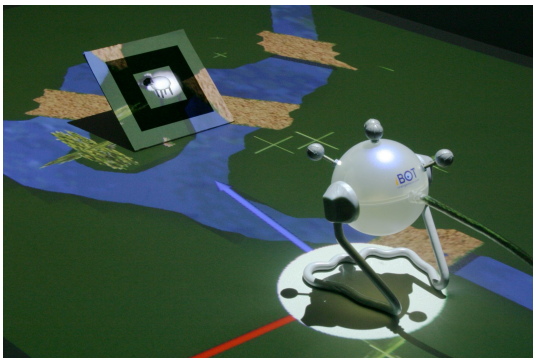

Figure 1. An ARToolkit marker tracked by a mobile camera attached to an ART target. A virtual sheep is projected onto the ARToolkit marker relative to the ART coordinate system.

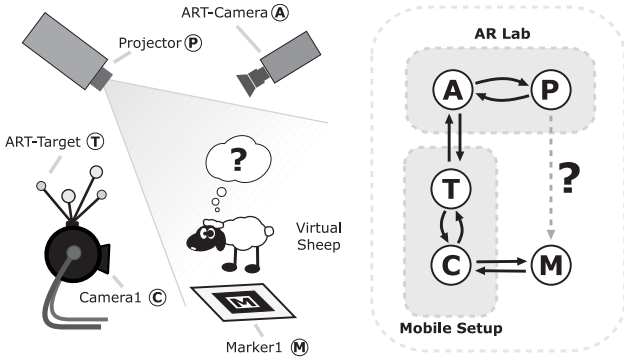

Figure 2. The example setup and its spatial relationship graph.

of either sensor taken individually. Figure 1 shows a camera tracked by an ART tracking system. Figure 2 shows both a schematic of the hardware setup, and a graph describing the corresponding spatial relationships. A ceiling-mounted projector P displays a pastoral landscape, populated by a sheep, on a table. The projector is calibrated such that it shares the same coordinate system as the ART tracker A; therefore, a static relationship exists between P and A. A mobile user is equipped with an ARToolkit [13] camera C on which an ART target T is mounted, resulting in another static relationship described by another edge in the graph. The camera is attached to a user's notebook with a wireless network interface. The middleware is running on the lab computers and on the notebook.

When the user enters the room, the two Ubitrack systems connect, and the description of an ARToolkit[1] marker is transferred to the user's notebook enabling the camera to track it. As long as the marker remains in the view of the camera, and the ART target attached to the camera is tracked by the ART system, then the virtual sheep can still be tracked in the coordinate frame of the ART system, even if it is physically out of range. Consequently, when the marker is moved the image of the sheep displayed by the projector can be seen to move accordingly. If the sheep is moved into a pen outside the range of the projector, it can still be viewed using some other tracked or fixed display.

The combination of these two tracking technologies is not novel in and of itself; however, this example focusses on the inference of spatial relationships and the resultant spontaneous behaviour of the system as it reacts by combining tracking sensors.

2 Formal Model

The goal of Ubiquitous Tracking is to provide, at any moment in time, an optimal estimate of the geometric relationships between arbitrary objects. An overview of the formal model is provided here, whilst further detail can be found in previous work [21, 28].

Spatial Relationship Graph Spatial relationships can be represented by a graph [3], in which objects are nodes, and spatial relationships between objects are directed edges. Each edge represents the spatial transformation between objects. A complete spatial relationship (SR) graph would represent environmental state in its entirety and could be used to query relationships between two arbitrary objects.

We can only make *estimates* of geometric relationships between real objects by taking *measurements*. Each measurement is made at a discrete point in time, yielding a geometric relationship that corresponds to the real relationship, but is corrupted by noise. The quality of a measurement is described using a set of *attributes* which includes properties such as latency, confidence values, or a standard deviation in metres. A *tracker* is a *sensor* that takes measurements of the spatial relationship between itself and other objects or *locatables*. Thus, edges are added to the graph or attributes updated in existing edges.

Example There are four objects of interest in the optical shared tracking scenario [15] shown in figure 3, namely two markers and two cameras. When viewed by an omniscient observer, the real spatial relationships between these objects are known at all times with perfect accuracy. A graph depicting this situation would be complete and all edges have attributes determined solely by known spatial relationships.

Camera A detects marker B, whilst camera D detects markers B and C. The associated graph still has four nodes,

[1] It is necessary to stress that the Ubitrack paradigm applies to *all* sensors, not just ARToolkit or other fiducial-based trackers.

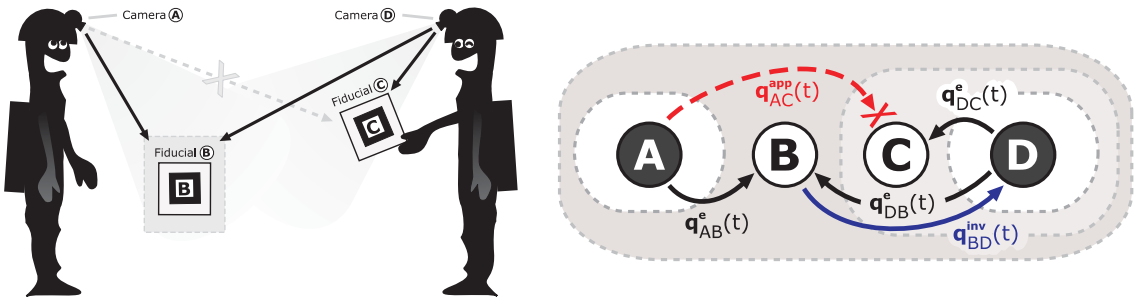

Figure 3. Example setup: Both cameras A and D detect fiducial marker B, but only camera D detects marker C. However, the application is interested in marker C's geometric relation to camera A. On the right, the graph containing all relevant inferred relations, as functions over time, is shown. Using this inferred knowledge, we can compute the desired relation between the objects A and C.

but is far from being complete, having only three directed edges AB, DB and DC. Note that the attribute values of these edges may differ significantly, as marker B may be well illuminated and therefore detected reliably, while marker C could be barely recognisable, leading to a low confidence attribute associated with edge DC.

Inference The goal is to obtain, at any point in time, an estimate of the spatial relationship between camera A and marker C, i.e. the pose associated with edge AC. We *infer* knowledge from the measurements and every inference results in a new edge in the graph. In figure 3 it can be seen that the pose of AC can be calculated using those of AB, DB and DC. Note that two measurements can only be directly combined if they were made at exactly the same time. A motion model and other knowledge is used to infer, new edges AB', DB' and DC' are created and their poses calculated over a continuous time domain, and their attributes describing the quality of these inferred relations are degraded to reflect their poorer quality compared to those of the discrete measurements. The next step is to infer the pose of BD' from DB' by inverting the spatial relationship. If the pose is represented using a 4×4 homogeneous matrix, the matrix is just inverted. Additionally, an appropriate attribute set for the edge BD' must be calculated, which may differ significantly from that of DB'. When pose is represented by homogeneous matrices, the target geometric relationship AC can be determined by multiplying the matrices from edges AB', BD' and DC'. Similarly, the attribute set of AC is obtained by propagating the attributes along the same path.

Evaluation Function Besides describing the characteristics of inferred geometric relationships, the main function of relationship attributes lies in resolving multiple solutions to a query. To this end, an evaluation function is defined that maps attributes along a path in the graph corresponding to a potential solution onto a real non-negative value. This value serves as a distance metric and is used to discriminate between alternative paths in the SR graph.

Error Model Each measurement describes a transformation from one coordinate system into another:

$$\mathbf{x}_{\text{new}} = \mathbf{t} + r\mathbf{x}r^*$$

$\mathbf{t}$ denotes the translation component, r the rotation quaternion and r^* its conjugated counterpart. All multiplications are quaternion multiplications. Adding noise to both position and orientation results in the following equation:

$$\mathbf{x}_{\text{new}} = \mathbf{t} + \mathbf{e}_t + re_r\mathbf{x}e_r^*r^*$$

where $\mathbf{e}_t = (e_{t_x}, e_{t_y}, e_{t_z})^T$ is the positional error and e_r is a quaternion representing a small rotation that can be approximated as: $e_r \approx (e_{r_x}, e_{r_y}, e_{r_z}, 1)$

A 6×6 covariance matrix C, associated with each measurement, describes the joint normal distribution of positional and orientational errors:

$$(e_{t_x}, e_{t_y}, e_{t_z}, e_{r_x}, e_{r_y}, e_{r_z})^T \sim N(\mathbf{0}, C)$$

To infer arbitrary spatial relations in the spatial relationship graph, multiple coordinate systems are concatenated, while maintaining appropriate error statistics. The concatenation can thus be expressed:

$$\mathbf{x}_{\text{new}} = \mathbf{t}_{\text{new}} + \mathbf{e}_{t_{\text{new}}} + r_{\text{new}}e_{r_{\text{new}}}\mathbf{x}e^*_{r_{\text{new}}}r^*_{\text{new}}$$

where

$$\begin{aligned}
\mathbf{t}_{\text{new}} &= \mathbf{t}_1 + r_1\mathbf{t}_2r_1^* \\
\mathbf{e}_{t_{\text{new}}} &= \mathbf{e}_{t_1} + r_1(e_{r_1}(\mathbf{t}_2 + \mathbf{e}_{t_2})e^*_{r_1} - \mathbf{t}_2)r_1^* \\
r_{\text{new}} &= r_1r_2 \\
e_{r_{\text{new}}} &= r_2^*e_{r_1}r_2e_{r_2}
\end{aligned}$$

This formula expresses how the combined positional error $\mathbf{e}_{t_{\text{new}}}$ depends not only on $\mathbf{e}_{t_1}$ and $\mathbf{e}_{t_2}$, but also on the rotational error e_{r_1} and the position $\mathbf{t}_2$.

In order to obtain a combined covariance matrix C, we compute the Jacobians $J_1 = \partial\mathbf{g}/\partial(\mathbf{e}_{t_1}, e_{r_1})^T$ and $J_2 = \partial\mathbf{g}/\partial(\mathbf{e}_{t_2}, e_{r_2})^T$ of $\mathbf{g}(\mathbf{e}_{t_1}, e_{r_1}, \mathbf{e}_{t_2}, e_{r_2}) = (\mathbf{e}_{t_{\text{new}}}, \mathbf{e}_{r_{\text{new}}})^T$ evaluated at $(\mathbf{e}_{t_1}, e_{r_1}, \mathbf{e}_{t_2}, e_{r_2})^T = \mathbf{0}$ and apply the error propagation law [14]:

$$C = J_1 C_1 J_1^T + J_2 C_2 J_2^T$$

This approach is similar to that of Hoff [11]. It is possible to apply this computation repeatedly for paths consisting of more than two edges.

Application of Error Model As mentioned above, when an application query could be resolved by multiple paths through the SR graph, the one that best fulfills the application's request should be chosen. For this purpose, an application-specified *evaluation function* is evaluated for each path and the path yielding the lowest value is returned.

In the current implementation, the covariance matrix for each path is explicitly computed using an average of the most recent measurements. The evaluation function then takes the trace of the positional part of the matrix. Other suitable evaluation functions exist, and we anticipate that their specification and efficient implementation will be the topic of future research.

The concepts of attributes and evaluation functions can also be exploited by arbitrary filter schemes combining data from multiple sensors. Each filter introduces new edges in which tradeoffs (e.g. increased latency and improved accuracy) are reflected in new attribute sets.

Modelling Complex Behavior More complex sensor behaviour can also be incorporated. For example, inertial trackers are treated as drifting orientation sensors. This drift relationship is modelled as yet another edge between the inertial tracker and world origin. Initially the extent of tracker drift is unknown, but whenever an alternative path through the graph from the inertial tracker to the world origin is determined, such as when the bearer is detected by another tracking system with an absolute reference frame, the drift can be corrected. An error model determines how drift uncertainty increases between corrections.

3 Implementation Concepts

An implementation of a Ubitrack system is proposed that can efficiently provide applications with estimates of spatial relationships in response to queries.

Data Flow Graphs At runtime, the Ubitrack framework builds a data flow graph that is distinct from the spatial relationship (SR) graph. The SR graph is an abstract model of the knowledge we have, or can infer, and is independent of the specific Ubitrack implementation. In contrast, data flow graphs are constructed at runtime by the Ubitrack implementation and combine tracker data, that is in constant flux, such that spatial relationships requested by an application can be delivered.

We have the key assumption that both the SR graph's topology and the attributes describing measurements' and inferences' qualities change less frequently than the spatial data. Thus, we can set up a static and efficient data flow graph once a path search in the graph has been performed and only occasionally have to double check the path's optimality.

Layered Architecture The Ubitrack architecture consists of three layers that are independent of the underlying implementation.

Sensor Layer incorporates all hardware devices and software components that deliver raw spatial data. A generic *Sensor API* provides an abstraction from the specific hardware devices while providing sufficient information to make full use of existing AR trackers.

Ubitrack Layer represents the formal model described earlier: aggregating tracking data, inferring knowledge of spatial relationships, and building runtime data flow graphs from the abstract spatial relationship graph. A *Query API* incorporating the desired spatial relationship defined by a source and target object, and the specification of the evaluation function is provided to applications.

Application Layer contains components that need spatial information, such as interaction controllers or displays. By accessing the Ubitrack layer, applications are relieved from details of sensor fusion or which trackers to choose, as everything is handled transparently by the Ubitrack layer.

3.1 Mapping onto DWARF

New users typically introduce their own equipment into the Ubitrack environment, which should interact with the whole AR environment without manual intervention. This equipment usually consists of wearable computers running AR applications, tracking devices and locatables. The integration of these components by the Ubitrack system should occur spontaneously, shielding the developer from the details of specific tracking technologies. Hence, DWARF was chosen to provide the middleware for dynamic integration of distributed components.

DWARF Basics Every DWARF system consists of network-transparent distributed components called *services*. Each service has *abilities* and *needs* stored in *service descriptions*. An ability describes the type of information that can be delivered and which communication protocol is used. Additionally, each ability can have arbitrary *attributes*[2] that describe characteristics of the data. Needs can have *predicates* that constrain the range of possible partners to those whose DWARF attributes have desired values. DWARF services communicate via remote CORBA method calls, and an event-based CORBA Notification Service, among other protocols. On each host there is a single *service manager* that collects and exchanges service descriptions between network hosts. When a match is found between a need and an ability, the desired communication channel is established by the service manager and services are started. This leads to an ad-hoc formation of chains of interdependent services that contribute to the application as a whole.

Mapping Ubitrack queries are issued by an AR application, and modelled as DWARF needs, of type *PoseData*, with DWARF attributes that specify the desired source and target nodes within the SR graph. An evaluation function, and *connectors* describing the desired communication protocol are also specified. Currently, an asynchronous push mechanism based on CORBA events is used in our tracking scenarios; however, other interfaces for synchronous and asynchronous pull mechanisms, as well as integration with OpenTracker, are planned. A new distributed middleware component, the *Ubitrack Middleware Agent* (UMA) described in section 3.2), searches the spatial relationship graph for the source and target node. Both nodes and edges of the SR graph are modelled as DWARF services.

- Nodes in the SR graph correspond to either active sensors such as cameras, or passive locatables such as markers and targets. Each object has a unique object identifier and is represented by *hardware services* that store these identifiers together with additional configuration data in their service descriptions (e.g. camera lense focal lengths or ARToolkit marker patterns).
- Edges in the SR graph are software components that provide the actual spatial measurements. These *tracking services* have abilities of type PoseData. The ability attributes contain the object identifiers of the source and target node of the SR graph edge.

Merging SR graphs When a mobile client is connected to a Ubitrack environment at the network level, the two systems exchange a description of their respective hardware services and reconfigure their tracking services accordingly [27]. The reconfigured mobile setup may now track new objects and update its SR graph with the new nodes. The edges connecting these new nodes correspond to new measurements.

If both setups have common locatables with identical object identifiers in their respective SR graphs, then the nodes corresponding to these locatables can be merged resulting in a single larger graph. This behaviour is demonstrated in the example described in the next section.

If no hardware descriptions of locatables are available, such as when performing markerless tracking, the mobile system adds *anonymous object nodes* into the SR graph. The two graphs cannot be merged, as the object identifiers are distinct. However, the mobile client can query its own Ubitrack instance to obtain all objects which are at or near the position of the anonymous one. If an object is found the identifier is replaced and the graph updated accordingly. This mechanism only works if the object has already been included in the SR graph.

If no result is returned to a query, the Ubitrack framework attempts to discover correlations between objects by monitoring the motion behaviour of potential candidates. By using an approach based on Lester et al. [16], velocity patterns of objects are compared. Objects that have different identifiers but nevertheless exhibit similar patterns of movement, are likely to correspond to the same object. Intermediate results from this approach are being evaluated.

3.2 Ubitrack Middleware Agent

The *Ubitrack Middleware Agent* (UMA) focusses on three aspects of the Ubitrack concept:

- Aggregation of object information and their interrelated measurements in order to build a distributed representation of the spatial relationship graph.
- Distributed path search in the spatial relationship graph depending on the properties specified in the application query.
- Dataflow setup for providing requested spatial information to an application.

SR Graph Modelling Each host in the network has a single UMA which periodically updates the local spatial relationship subgraph. Nodes in the SR graph represent real or virtual objects, and thus are associated with the computer where information about them is stored. Edges in the SR graph are associated with the abilities of local tracking services to provide spatial relationships. Changes in the attribute set of the measurements are updated periodically by

[2] these are distinct from spatial relationship edge attributes described in section 2, although SR graph edge attributes are mapped onto ability attributes

the tracking services, as changes to their abilities' attributes, and propagated to the local UMA.

An inference process determines appropriate inverse relationships for specific edges, as inverse edges may have significantly different attributes as explained in section 2. This leads to a new set of inferred relationships, that result in an undirected SR graph.

Scalability Most tracking events will be generated and consumed by devices that are physically in close proximity to one another, that is, in the same *locale*. Thus, in general, devices whose nodes are close together in the distributed SR graph will also be logically close to one another at the network level. Hence, network communication can be reduced and spatial cues for integration of mobile and stationary clients exploited. The full range of AR and Ubicomp applications can be facilitated by also supporting global queries. For example a "God's eye view" of the world in the form of a map-browser would not require the high-performance associated with a traditional AR application, but would involve communication with every world object, or the entities responsible for them.

Path Search In order to execute a distributed path search, edges between arbitrary objects stored by two different UMAs require a direct connection between them in the form of a bidirectional communication channel. Therefore a UMA has an ability to provide an event channel for each local object (tracker or locatable). When an edge is found whose target object is not one of the local objects, a need for an appropriate event channel emerges. Matching needs and abilities are connected by the DWARF middleware.

The path search between the UMAs is implemented using a distributed asynchronous Bellman-Ford algorithm [18]. Each new search request specifies four quantities: a search identifier, the source node, the target node, and an evaluation function that computes edge weights from attribute sets. If the evaluation function is evaluated along the path from the source node to another node, we obtain a cost or *distance* value on which Bellman-Ford is based. When a node is visited during a path search, the distance of this node from the source node is updated. Each node must know the distance of each of its neighbors' minimum-cost path to the source node. Hence, when a search computes a new minimum distance between a node and a source node, the node's neighbours must be informed of this value. Therefore a participation identifier is stored additionally for each distance update that is sent out. When a search event is received by the UMA, the distance associated with the current path is compared with those from earlier participations. If the new distance is less than that of the current candidate path, then paths to all outgoing edges are computed and new search events are sent to relevant UMAs. If the distance is greater than that of the current or target node is found, an acknowledgement event is sent back. Each time a UMA participates in a path search, it must wait for acknowledgement events or a timeout before sending back acknowledgement events to the source. After all search messages, sent out by the UMAs containing the source node, have been acknowledged, the UMA chooses the path that best matches the requirements specified in the application query using the evaluation function. The UMA then configures the data flow graph accordingly by creating new DWARF inference services and letting them connect them to the running tracking services, as shown in figure 4.

In order to combine measurements along a path in the spatial relationship graph, the UMA instantiates a data flow graph that performs the actual computations. As the authors have been working on both the DWARF and *Studierstube* [24] AR frameworks, two data flow graph architectures were designed to fit on top of their respective frameworks and enabling straightforward integration of Ubitrack concepts into existing setups. Future work will include further integration of these two approaches.

DWARF Data Flow Inference components are realised as individual services each with a single ability for sending combined measurements to applications, and an arbitrary number of needs. The needs and abilities are automatically set by the UMA in order to connect tracking services that form the chosen path.

Thus far, there has been an implicit assumption that all measurements occur simultaneously, which is rarely the case in real systems. To solve this problem, inference components perform interpolation and extrapolation to compute valid estimates for specified moments in time. In order to calculate position, a simple linear interpolator is used, while orientation is calculated using the SLERP algorithm [6]. Extrapolation results in greater uncertainty in position and orientation. In future implementations, we plan to include Kalman filters with different motion models for each tracking device.

OpenTracker Data Flow The original OpenTracker library [23] implements the well known "pipes & filters" dataflow pattern, in which pose data is modelled as a sequence of synchronous objects of the same time referential that is then transported across a network of interlinked processing filters. In order to support the dynamic dataflow requirements of Ubitrack, we extended OpenTracker by specifying a runtime reconfiguration API. Every instance of an OpenTracker network is governed by a UMA that issues explicit reconfiguration commands to create, delete or modify dataflow paths within the network. Results from spatial relationship queries always yield the addressable instance of participating sensors, which enables the UMA

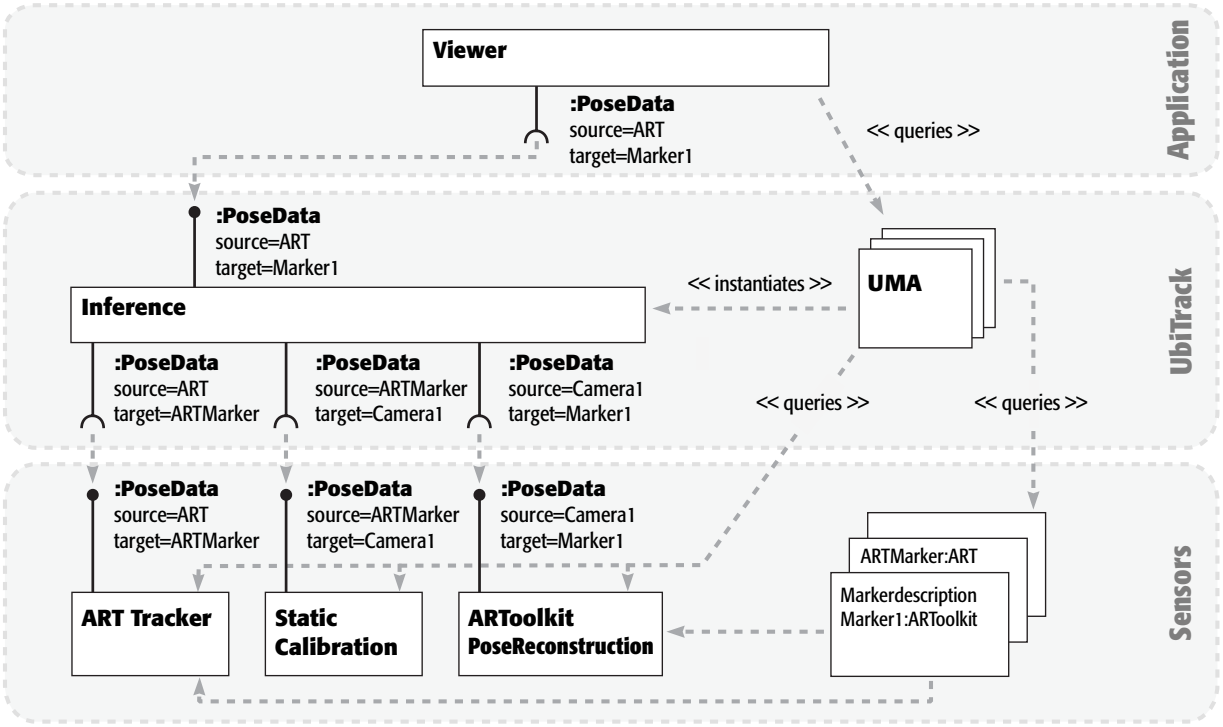

Figure 4. DWARF services structured in the three layer Ubitrack model.

to set up peer-to-peer communication channels between OpenTracker contexts to acquire and cache remote measurements. The process of inverting and interpolating measurements is handled by a set of specialised filters that are inserted into dataflow paths by the UMA as required.

Comparison between both data flow systems OpenTracker dataflow paths are periodically pre-computed, and consist of peer-to-peer event streaming channels and a sequence of single-threaded processing functions. The details of OpenTracker dataflow are explicity managed by a UMA, whereas DWARF services implicitly establish CORBA communication channels based on their needs and abilities. Due to its underlying middleware, DWARF cannot adapt to environmental change as rapidly as OpenTracker, but behaves more spontaneously requiring less intervention on the part of UMA components during the dataflow reconfiguration procedure.

4 Simulation Environment

Only a small number of people and trackers can feasibly participate in repeatable large-scale experiments due to resource constraints. Consequently, we have developed a Ubitrack simulator that makes it possible to test the contribution of specific tracking technologies in different configurations before purchasing, deploying or extending a Ubitrack environment. Furthermore, it is possible to experiment with trackers that are currently unavailable such as scarce, or nascent positioning technologies based on ultrasound [10] or ultra-wideband [26] (UWB) signals that deliver 3 DOF position estimates of potentially thousands of objects throughout entire buildings. This base level of tracking coverage enables diverse Sentient Computing [1] applications as well as the introduction of many new artifacts into the AR environment.

The simulator was designed such that distributed processes modelling different sensors can connect to the simulation engine and generate events. Figure 5 shows a screenshot from a GUI tool that can determine the paths and speed of virtual people throughout a building using splines. Each person is equipped with different sensors, trackers and locatables. The interactions of these personal devices with one another and the building infrastructure can be observed.

Active tracker regions are cross-hatched and coloured according to whether locatables are detectable or not. Similarly, fiducial markers attached to the walls and other locatables can also change colour when detected. The envelopes of static trackers are displayed as rectangles while camera frustums are displayed as triangular regions. Figure 7 shows a 3D visualisation, corresponding to the viewpoints of these cameras, that can be used to send images to be analysed by an ARToolkit instance to generate tracking events.

5 Results

The implementation of the example described in section 1 uses two computers: one stationary system running an ARTTracker service, and the other a mobile system running the ARToolkit service. The ARTTracker service has an ability measuring the relationship from the ART DTrack coordinate system to the ARTMarker1 object. The video output of this computer is connected to a projector, showing an image on a table. An ARToolkit marker description service provides the configuration necessary to detect the marker on the table.

In addition to running the ARToolkit service, the mobile system also stores information concerning the fixed geometric relationship between the ART target and the camera coordinate frame. When the mobile computer enters the AR lab, its ARToolkit tracker is reconfigured such that it can create a new ability for the measurements between the camera and the marker [27].

The UMA services on both the stationary and the mobile systems query locally available abilities in order to build the local spatial relationship subgraph. This information is then exchanged to construct the final graph depicted in figure 2.

In order to render a virtual sheep onto the ARToolkit marker, we start a 3D Viewer service on the stationary computer, with a need for the relation between the ART tracker and the Marker1 object. The local UMA triggers a distributed path search and finally instantiates a new inference component that computes the combined measurements and delivers them to the Viewer. The virtual sheep on the marker is shown in figure 1.

Simulation Environment Figure 5 shows four individuals: Bonnie, Clyde, Jack and Jill. Bonnie, Clyde and Jill all have cameras, and all four are depicted at a moment when they are inside the active regions of fixed trackers. Figure 6 shows the instantaneous SR graph, obtained from the UMA, resulting from a query of the spatial relationship of all four individuals with respect to the world origin "Root". Figure 7 shows an image from the camera worn by Bonnie. A closer study of the spatial relationship graph in figure 6 shows that this camera "firefly4" is being used to calculate orientation relative to the ARToolkit marker "floor4_94". Bonnie's position is being determined by the "HallwayUWBTracker" which is only a 3 DOF tracker. Note that, although each person has similar equipment, different solutions have been inferred from their respective contexts.

6 Future Work

We anticipate a genuine commercial demand for middleware that can seamlessly integrate tracking hotspots, with

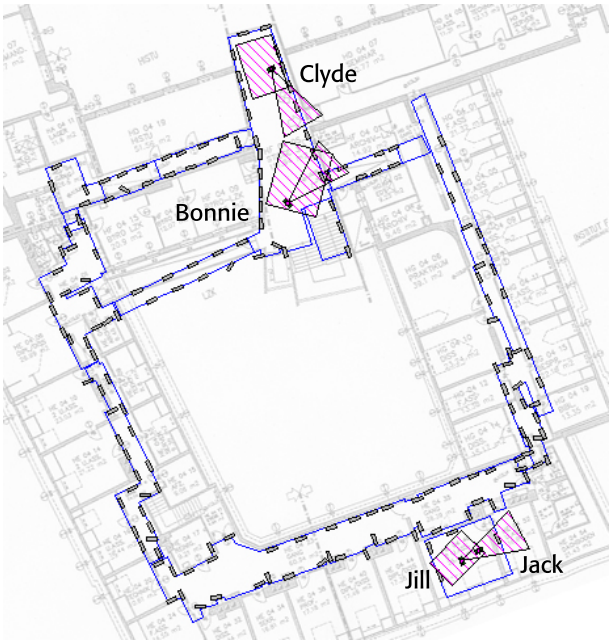

Figure 5. Simulation Environment

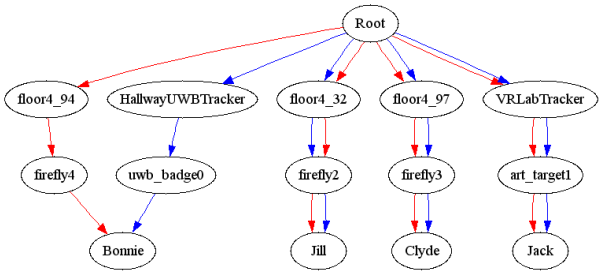

Figure 6. Spatial Relationship Graph

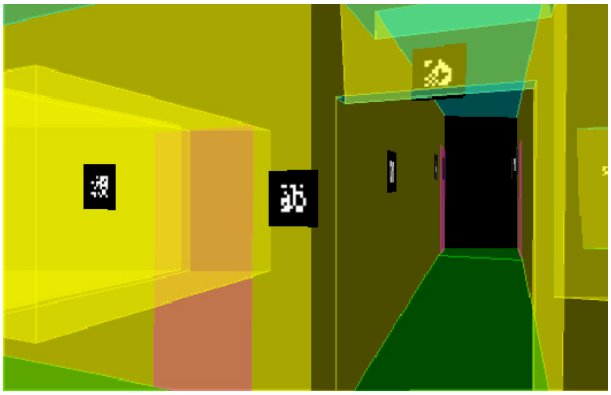

Figure 7. Image from camera worn by the character, Bonnie, shown in Figure 5

personal sensors and wide-area trackers. The Ubitrack framework can deliver the tracking quality of service where a client actually *needs* it, using the equipment the client can afford. We are currently working with industrial partners to investigate the deployment of the Ubitrack framework in real settings such as interactive collaborative museum exhibits, and airport security.

Sensors Currently, the bulk of sensors supported by Ubitrack are limited to a handful of devices which all measure position and/or orientation. However, there are many other sensors that provide useful information concerning environmental state.

To accomodate more general measurements, we will extend the Ubitrack formalism in two ways. First, we will replace the current state space model that can be represented by a 4×4 homogeneous matrix by something more general. Further research is required concerning which sets of attributes can be used to describe measurements, e.g. by parameterising non-gaussian error models. We would then be able to use other sensors commonly used in the Ubicomp community. Accelerometers are very cheap, low power and increasingly embedded in mobile devices. Cell-based trackers such as the Active Badge [29] commonly track to the granularity of a single room and can be modelled as position measurements whose error distributions are uniformly spatially distributed throughout the room. Similarly, RFID tags can indicate proximity of a tag worn by a user at a given point in time. Range sensors can be based on many quantities such as time-of-flight, time-of-arrival, or received signal strength of ultrasonic, 802.11 or UMTS signals.

Computer Vision techniques can be used to determine pose based on natural features in an uninstrumented environment. Consequently, these trackers have the greatest potential to affordably increase the quality and range of AR environments. However, these trackers are currently very brittle, as they will lose track when unable to detect any suitable features (e.g. when looking at a monochrome wall). By using the Ubitrack framework as a bootstrapping mechanism it should be possible to integrate these trackers such that they can be reasonably deployed in real environments.

Environmental Constraints Given knowledge of the environment such as the locations of walls and other obstacles, it is possible to create new inferences that exploit the constraint that objects cannot generally pass through walls. Hightower et al. [8] employ Bayesian filters, particle filters and Voronoi graphs [17], while Höllerer et al. [12] use a combination of spatial maps and accessibility graphs. These techniques should be incorporated into the Ubitrack Layer so that inferences can be made on the basis of environmental knowledge, not solely on measurements.

Autocalibration The process of deploying and maintaining sensor infrastructure is very time-consuming, tedious and expensive. For example, when a sensor is installed or accidentally moved, the Ubitrack framework should use redundant tracking information to check that the sensor is calibrated in a way that is consistent with its environmental knowledge. This is similar to the treatment of inertial drift in section 2. The calibration should then either be changed automatically, or a technician alerted.

7 Conclusion

We have developed a formal model for Ubiquitous Tracking that allows us to construct graphs, from distributed information sources, of 6 DOF spatial relationships and their attributes. We have tested an initial implementation, in a small laboratory setup, combining two different optical trackers. Hence, the fundamental feasibility of our approach has been demonstrated. To test the scalability of this approach we have developed a simulation environment for modelling building-scale AR scenarios.

Important future work includes the extension of the model to accommodate other types of sensors, and deployment in large-scale environments.

The Ubitrack approach to integration of arbitrary tracking sensors promises greater user mobility in rich AR environments capable of scaling to building, campus or city orders of magnitude. The dynamic nature of the middleware allows adaption to changes in infrastructure and users' devices, while optimally meeting varying quality-of-service demands of diverse AR and Ubicomp applications.

Ubiquitous Tracking can make large-scale tracking feasible, both technically and economically, thus greatly enhancing the utility of Augmented Reality.

Acknowledgements

This work was partially supported by the Deutsche Forschungsgemeinschaft, project DySenNetz (KL1460/1-1), the Austrian Science Foundation under contract no. Y193, the Vienna University of Technology, Forschungsinfrastrukturvorhaben TUWP16/2002 and EU contract IST-28610-2000.

References

[1] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, and A. Hopper. Implementing a sentient computing system. *IEEE Computer*, 34(8):50–56, 2001.

[2] 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.

[3] B. Brumitt, J. Krumm, B. Meyers, and S. Shafer. Ubiquitous computing and the role of geometry. *IEEE Personal Communications*, pages 41–43, October 2000.

[4] E. M. Coelho and B. MacIntyre. High-level tracker abstractions for augmented reality system design. In *International Workshop on Software Technology for Augmented Reality Systems*, pages 12–15, October 2003.

[5] D. Curtis, D. Mizell, P. Gruenbaum, and A. Janin. Several devils in the details: Making an AR app work in the airplane factory. In *First IEEE Workshop on Augmented Reality*, November 1998.

[6] E. B. Dam, M. Koch, and M. Lillholm. Quaternions, interpolation and animation. Technical Report DIKU-TR-98/5, University of Copenhagen, July 1998.

[7] S. Feiner, B. MacIntyre, T. Höllerer, and T. Webster. A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. In *Proc. of ISWC 1997*, October 1997.

[8] D. Fox, J. Hightower, H. Kautz, L. Liao, and D. Patterson. Bayesian techniques for location estimation. In *Proceedings of The 2003 Workshop on Location-Aware Computing*, October 2003.

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

[10] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. In *Mobile Computing and Networking*, pages 59–68, 1999.

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

[12] T. Höllerer, D. Hallaway, N. Tinna, and S. Feiner. Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system. In *2nd International Workshop on Artificial Intelligence in Mobile Systems (AIMS '01)*, pages 31–37, 2001.

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

[14] K.-R. Koch. *Parameterschätzung und Hypothesentests in linearen Modellen*. Dümmler, Bonn, 1980.

[15] F. Ledermann, G. Reitmayr, and D. Schmalstieg. Dynamically shared optical tracking. In *Proceedings of the First Augmented Reality Toolkit Workshop*, 2002.

[16] J. Lester, B. Hannaford, and G. Borriello. "Are You with Me?" – using accelerometers to determine if two devices are carried by the same person. In *Pervasive Computing: Second International Conference*, pages 33–50, Linz/Vienna, Austria, 2004.

[17] L. Liao, D. Fox, J. Hightower, H. Kautz, and D. Schulz. Voronoi tracking: Location estimation using sparse and noisy sensor data. In *Proc. of Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE/RSJ, 2003.

[18] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc, 1996.

[19] B. MacIntyre, E. M. Coelho, and S. Julier. Estimating and adapting to registration errors in augmented reality systems. In *IEEE Virtual Reality Conference 2002 (VR 2002)*, Orlando, Florida, March 2002.

[20] J. Newman, D. Ingram, and A. 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, October 2001.

[21] J. Newman, M. Wagner, T. Pintaric, A. MacWilliams, M. Bauer, G. Klinker, and D. Schmalstieg. Fundamentals of ubiquitous tracking for augmented reality. Technical Report TR-188-2-2003-34, Vienna University of Technology, 2003.

[22] W. Piekarski and B. 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.

[23] G. Reitmayr and D. Schmalstieg. OpenTracker – An Open Software Architecture for Reconfigurable Tracking based on XML. In *Proceedings of the ACM Symposium on Virtual Reality Software & Technology (VRST)*, Banff, Alberta, Canada, 2001.

[24] D. Schmalstieg, A. Fuhrmann, G. Hesina, Z. Szalavari, L. M. Encarnacao, M. Gervautz, and W. Purgathofer. The Studierstube augmented reality project. *PRESENCE - Teleoperators and Virtual Environments*, 11(1):32–45, 2002.

[25] R. M. Taylor, ll, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. VRPN: a device-independent, network-transparent VR peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 55–61. ACM Press, 2001.

[26] Ubisense. Ubisense limited. http://www.ubisense.net/ [2004, May 13], 2004.

[27] M. Wagner and G. Klinker. An architecture for distributed spatial configuration of context aware applications. In *2nd International Conference on Mobile and Ubiquitous Multimedia*, Norrköping, Sweden, 2003.

[28] M. Wagner, A. MacWilliams, M. Bauer, G. Klinker, J. Newman, T. Pintaric, and D. Schmalstieg. Fundamentals of ubiquitous tracking. In *Advances in Pervasive Computing*, pages 285–290. Austrian Computer Society, 2004.

[29] R. Want, A. Hopper, V. Falcão, and J. Gibbons. The active badge location system. Technical Report 92.1, Olivetti Research Ltd. (ORL), Cambridge, 1992.

[30] M. Weiser and J. S. Brown. *Beyond Calculation: The Next Fifty Years of Computing*. Copernicus, 1998.