A System Architecture for Ubiquitous Tracking Environments

Manuel Huber Daniel Pustka Peter Keitler Florian Echtler Gudrun Klinker*

Technische Universität München
Fakultät für Informatik
Boltzmannstraße 3
85748 Garching b. München, Germany

ABSTRACT

Ubiquitous tracking setups, covering large tracking areas with many heterogeneous sensors of varying accuracy, require dedicated middleware to facilitate development of stationary and mobile applications by providing a simple interface and encapsulating the details of sensing, calibration and sensor fusion.

In this paper we present a centrally coordinated peer-to-peer architecture for ubiquitous tracking, where a server computes optimal data flow configurations for sensor and application clients, which are directly exchanging tracking data with low latency using a lightweight data flow framework. The server's decisions are inferred from an actively maintained central spatial relationship graph using spatial relationship patterns.

The system is compared to a previous Ubitrack implementation using the highly distributed DWARF middleware. It exhibits significantly better performance in a reference scenario.

Index Terms: I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Tracking H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

1 MOTIVATION

In industrial augmented reality scenarios, there is a growing demand for integrated working environments which span large factory buildings. In such an environment, many different mobile and stationary AR-supported applications, such as logistics, production, maintenance or factory planning may coexist and require shared access to permanent tracking with varying accuracy requirements. Today, no single technology exists that satisfies the tracking requirements of all these applications and can – at least for a reasonable price – be deployed throughout such an environment. For this reason, in a realistic setup, many different tracking systems would be installed ranging from low-precision wide-area WLAN tracking to infrared-optical systems covering only small areas with high accuracy. The installation, maintenance and expansion of such a large-scale heterogeneous tracking environment poses new challenges to the underlying middleware concepts.

Heterogeneous wide-area tracking environments Emerging tracking methods based on technologies like WLAN or RFID provide the possibility to deploy tracking to ever-enlarging indoor areas. With increasing tracker coverage, a larger diversity of AR applications will need to share this tracking infrastructure. Stationary applications that are already in use will more and more be complemented by mobile applications that would have been completely impossible without wide-area tracking. Also, applications that are stationary today, might benefit from enlarging tracking areas and become more adaptive and better integrated in the productive environment. Many of these wide-area tracking systems have the drawback of being rather imprecise. Nevertheless, they serve quite well for navigation problems and can thereby bridge the gap between islands of higher tracking accuracy. Furthermore, they can provide useful initial positions to other sensors, such as markerless optical trackers [7]. There are also many examples where a fusion of measurements from different mobile and stationary sensors improves overall tracking quality and robustness, e.g. [4].

Separation of tracking from applications Since multiple applications shall be able to use a common infrastructure, it is necessary to decouple this infrastructure from application development and operation. Existing AR applications are normally tightly coupled with the underlying tracking system. This means that administrative tasks relating to the tracking system are normally integrated in the application itself. Encapsulation of the tracking infrastructure in a middleware facilitates administrating and calibrating the environment independently of the applications. On the other hand, tracking systems produced by different manufacturers usually come with different drivers and APIs. A suitable tracking middleware has to provide the necessary abstractions from such implementation details. To allow an easy exchange of different types of tracking systems, the details about the numerous different coordinate frames involved also need to be hidden from the application developer. Seamless handover of mobile applications between different tracking areas requires in particular that this abstraction can be dynamically reconfigured at runtime.

1.1 Proposed Approach

Ubiquitous tracking (Ubitrack) is a middleware concept for tracking. It supports distributed applications to use a common, heterogeneous sensor network. Our goal is not the invention of new sensors, nor is it the development of new algorithms for tracking, calibration or sensor fusion but rather the integration of these existing components into a single framework where an intelligent middleware automatically selects the best algorithms to transform and fuse tracking data, based on available sensors, application needs and client capabilities. In the ideal case, the Ubitrack system should perform as well as a hand-tuned system in providing accurate tracking data.

1.2 Related Work

Software to construct data flow networks for abstraction and processing of sensor data for AR and VR applications already exists, with the two most prominent examples being OpenTracker [12] and VRPN [13]. Such data flow networks constitute one of the most important building blocks of ubiquitous tracking environments and provide the necessary transformation and network transport of tracking data. However, to date, these software libraries need to be manually configured for each tracking situation and cannot handle dynamic changes, such as the addition of a new sensor at runtime.

In the pervasive computing community, many approaches exist to handle user context and/or location [3, 5, 2]. While, similar to the proposed approach, these systems include models for abstraction

*e-mail: { huberma, pustka, keitler, echtler, klinker }@in.tum.de

of sensor data, they focus on a much broader diversity of sensors than those used in typical AR/VR setups. As location in pervasive computing is usually needed to generate discrete location events, e.g. when somebody enters a building, low-latency data processing and transport is not considered critical.

In our previous work [8] we have successfully demonstrated a Ubitrack system that dynamically instantiates data flow components to convert 6DOF tracking information from various sources into a target coordinate system convenient for the application programmer. Unfortunately, the system did not have the required performance for larger setups in terms of processor load and reaction time to structural changes in the environment. This was mostly caused by the underlying DWARF middleware. A detailed comparison will be presented in Section 3. We also recently introduced the concept of spatial relationship patterns [10]. It provides the formal framework in which different algorithms for tracking, calibration and sensor fusion can be described and automatically selected, depending on the current sensing constellation. The reader is referred to [10] for details about these concepts as well as [9] which discusses related synchronization issues in generated dataflow networks.

2 UBITRACK SYSTEM ARCHITECTURE

In this section, the Ubitrack system architecture is outlined. The key aspects of the philosophy behind Ubitrack have already been described in Section 1. The requirements towards the architecture will be substantiated in the following before giving an overview of the chosen approach and describing the individual components in more detail.

Dynamic reconfigurations In a production environment where several applications are running in parallel using the same common tracking infrastructure, it is not admissible to shut down the whole environment for maintenance tasks such as the recalibration of an existing or the integration of a new sensor. Furthermore, in some scenarios, it is necessary to cope with a dynamically changing number of trackable objects and even with mobile applications (dis)appearing at runtime and moving between different trackers.

Efficient communication of tracking data at runtime Interposing a tracking middleware must not lead to a degradation in tracking quality. This especially means that sensor measurements have to be transported and processed as fast as possible so that there is no delay for the AR-application that would not also be existent if the middleware were not used.

Queries for involved objects and transformations Applications have to be able to query for available objects and their spatial relationships. This is a direct consequence of separating the tracking and application domains. If applications are developed without knowing the concrete layout of the underlying tracking environment, they have to express their needs regarding available objects and the spatial relationship(s) between them in some way.

Varying capabilities of the participating application platforms Especially for mobile AR applications running on PDAs, the available interfaces and also the computing power is often limited. In order to provide them with high-quality tracking resulting from different sensors and computationally expensive sensor fusion, the only solution may be to attach the sensors to another machine and also to source out some or all of the computations. The final results can then be transmitted via network and directly consumed by the visualization module on the mobile device.

2.1 Overview

With the mentioned requirements in mind, we decided to use a *centrally coordinated peer-to-peer architecture* for the Ubitrack middleware, availing from the advantages of both a pure client-server and a peer-to-peer architecture. Figure 1 gives an overview over the Ubitrack environment.

Compared to the pure peer-to-peer approach described in [8], the coordination and maintenance of a tracking environment is much easier using the client-server-approach. For the highly time-critical propagation of tracking data, however, we benefit from using the shortest communication paths between suppliers and consumers, avoiding latency due to the indirection via a server and, as a consequence, avoiding also the corresponding processing load for the server.

The individual components of the Ubitrack system architecture are described in more detail in the next sections.

2.2 Clients

A *client* in the Ubitrack environment is an entity that is involved in the flow of tracking data. It can run on a dedicated hardware platform, though this is not a requirement. For all functionality related to tracking, the *application* software running on the client relies on the *Ubitrack library*. The application uses the API of the library in order to register its base SRG(s) describing the spatial relationships between locally connected *sensors* and other real or virtual objects at the *Ubitrack server*. Furthermore, using the same mechanism, it also registers its requirements towards the tracking systems in terms of application queries. The server coordinates the requests of all clients and assigns a data flow network to each of them.

For the purpose of clarity, we outline two archetypical types of clients (see also Figure 1), although very often, the client will be a mixture of both.

Sensor client A sensor client provides tracking data to the rest of the system. For this purpose, it instantiates a data flow network which uses the measurements of the sensors it owns, potentially filters and/or fuses them and finally feeds the results into the peer-to-peer network so that they can be further processed in data flow networks running on other clients. The actual sensor hardware is treated as a black box interfacing with the Ubitrack library using vendor specific APIs.

Application client The application client is the exact counterpart of the sensor client. It is a pure consumer of tracking data and relies on the data flow networks of other clients to provide it with the data it needs. The most simple data flow network of an application client consists of just two data flow components, one receiving data from the peer-to-peer network and another handing it over to the application.

The clients themselves consist of different components.

Application The application software relies on the API of the Ubitrack library for all purposes related to tracking. Furthermore, it may perform other more or less complex tasks. A typical AR application for example would register a query for the transformation of some virtual objects to the HMD coordinate frame and use the tracking results to render them in the HMD.

Ubitrack Library The Ubitrack library encapsulates all communication between the client and the Ubitrack server and provides means to instantiate a data flow network according to the description constructed by the server.

Before the server can compute descriptions of suitable data flow networks, the client applications first have to announce their tracking abilities and needs via the API of the library. A pure sensor client on the one hand registers only SRG parts describing sensors, tracked objects and known spatial relationships between them. An application running on a pure application client on the other hand registers only queries describing the application's need(s) in terms of objects and spatial relationships satisfying certain predicates.

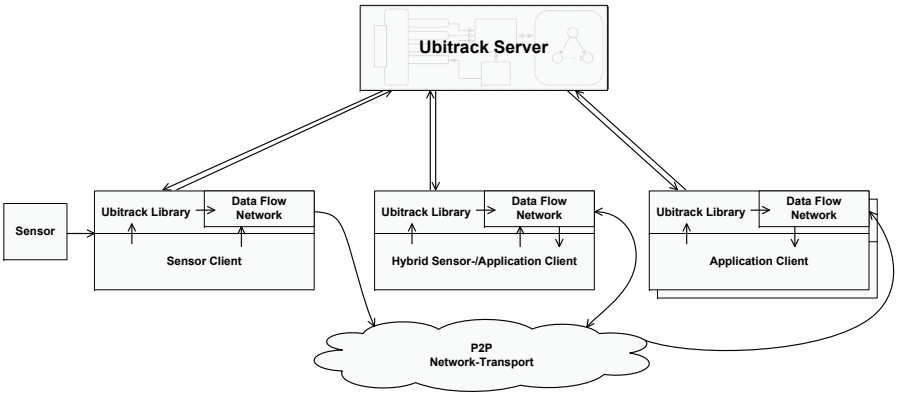

Figure 1: Ubitrack Architecture overview

Of course, the application can also change or delete SRG parts or queries previously registered at any time without restarting the system. SRG updates and queries are registered by the application via the API in the *Ubiquitous Tracking Query Language (UTQL)* format and are handed over to the server. The UTQL description is heavily based on the pattern abstraction of spatial relationship graph transformations to express registrations, client abilities and queries in a uniform way. See [11] for an in-depth discussion of UTQL.

The Ubitrack library also uses UTQL to register data flow components it is able to instantiate as part of its data flow network. Again this is done by describing their spatial relationship patterns and communicating a list of available patterns to the server. These lists of available SR patterns may differ from client to client due to differing library versions or computing power.

Dynamically Reconfigurable Data Flow Network The Ubitrack library provides a set of light-weight *data flow components* for the creation of data flow networks. The data flow components are dynamically instantiated and connected in order to establish a data flow network according to the description generated by the Ubitrack server. Each time the global data flow description is rearranged by the server, induced by some change in the list of registered SRGs, the affected clients rearrange their data flow network accordingly on the fly.

2.3 Ubitrack Server

The Ubitrack server is the central coordination unit of the system. It administrates a list of all base SRGs, application queries and SR patterns that may be registered and deregistered by the participating clients at runtime. All currently registered base SRGs together make up the *world SRG*. It represents the general knowledge about the tracking environment provided by the clients and changes whenever the list of registered base SRGs changes.

The Ubitrack server always tries to fulfill all registered queries. For this purpose it applies the registered SR patterns to the world SRG with the purpose of somehow deriving the SRG edge denoted by the query. As soon as the server succeeds in finding a sequence of pattern applications in order to derive a queried edge, it communicates this knowledge to the client the query came from. In case tracking data has to be provided by other clients via the peer-to-peer network in order to compute edges contained in the query, the server also updates the data flow description of the other clients. Thereby, it also considers the varying client capabilities.

It is important to state that the Ubitrack server itself does not know anything about the underlying data flow algorithms. Its derivations merely depend on the question about which constellation of edges in the SRG is necessary so that a certain SR pattern can be applied in order to provide a new edge.

2.4 Management Tools

Management tools are used for the configuration of a tracking environment in the design phase as well as for applying and monitoring dynamic reconfigurations at runtime. There are two main tasks which are assisted by these tools:

Design phase The design phase mainly comprises modeling the base SRGs that describe the transformations (edges) between different coordinate frames (nodes) in the world as it is seen by the different clients. For this purpose we have implemented a graphical editor to facilitate the process of creating new SRGs which can then be used by the corresponding client applications to register their abilities and needs towards the tracking environment.

Runtime While the tracking environment is in operation, an administration tool is able to provide insights into the internals of the Ubitrack server, namely the current list of registered base SRGs, SR patterns and application queries per client as well as the current world SRG and the data flow networks derived from it.

3 Results

To evaluate the system architecture, we implemented a simple scenario that allows us to gain insights into the performance of the system. The scenario is similar to the one presented in [6], but with a more complex spatial relationship graph.

The scenario consists of a pastoral landscape on which a herd of virtual sheep is grazing. In order to increase the complexity of the SRG, the sheep implement some herding behavior and therefore require the position of each other. The pasture with the herd of sheep is visualized inside an HMD, tracked by an infrared-optical outside-in tracking system (ART DTrack). To have more distinct coordinate frames in the setup, we also added a tangible plastic sheep, directly tracked by the outside-in tracker. An additional tangible sheep was placed on an AR-Toolkit-like square marker, tracked by a firewire camera. Its pose was again tracked by the outside-in tracker.

The visualization inside the HMD was produced using a separate rendering client, which sent out a single UTQL query for all renderable objects.

As the first qualitative result, the system was able to provide the coordinates of all renderable objects (landscape, virtual and tangible sheep) in the coordinate frame of the HMD, even though the objects were provided in different coordinate frames. The server also correctly inserted synchronization components (interpolation) into the data flow network where necessary. Additional sheep could be added to the system at any time.

At run-time, the system had a latency comparable to a hand-tuned implementation, because all the tracking computations for

visualization took place inside the rendering client using our light-weight data flow framework.

Comparison to DWARF For quantitative evaluation of the new architecture, we compared it to our first implementation of a Ubitrack system [8], based on the DWARF [1] middleware. To our knowledge, this is the only system with a comparable (although more limited) functionality. The most significant differences between the two implementations are:

The two implementations differ in a number of ways: First, the DWARF system is highly distributed without a central instance for coordination. Instead, computers broadcast their "needs and abilities" in a peer-to-peer fashion. Second, each DWARF service (comparable to a data flow component) runs in an independent operating system process. Communication between services is done using CORBA, with a high overhead of process switching. Finally, the generation of data flow graphs in DWARF is based on a path-search in graphs. This is roughly comparable to the pattern matching when only the inversion and concatenation patterns are used.

We implemented the scenario described above using both architectures and measured both response time with regard to SRG changes and system load on the rendering client.

Response Time The time from the addition of a new virtual sheep to the arrival of the first tracking data at the rendering client took an average of 13 seconds using the DWARF-based implementation whereas the centrally-coordinated architecture required only 35ms for a single sheep. When 10 sheep were added simultaneously, the DWARF implementation took 22s vs. 280ms. Furthermore, the DWARF system was not usable during reconfiguration as no tracking data was delivered. Setting up an SRG of 30 virtual sheep was impossible in DWARF, because of limitations in the underlying libraries and network protocols, but required only 1s in the new architecture. In this case, the extra time needed by the centrally coordinated peer-to-peer architecture was mostly consumed by the pattern detection. Thus, by further improvement of the algorithms, the total system performance can be improved without architectural changes.

Processor Load Data flow networks constructed using DWARF services consume significantly more processor time, because each service is a separate process, that uses CORBA for communication. In our experiments, the system already had a load of 100% with only 10 sheep posting their position at a frequency of 30Hz. In contrast, the centrally coordinated architecture, running 30 sheep, merely produced a processor load of about 4%. This example dramatically shows the importance a light-weight data flow framework with little communication overhead.

4 Conclusion and Future Work

The specification of tracking environments as a spatial relationship graph is a new way to deal with large-scale heterogeneous systems. By describing processing capabilities and queries as spatial relationship patterns, specified in UTQL, a large diversity of clients can coexist and exchange data in such an environment, without application developers needing to worry about the details of different trackers and coordinate systems.

Our experiments showed that the centrally coordinated peer-to-peer architecture is well suited to implementing ubiquitous tracking environments, as it combines the advantage of a quick response time of a centralized server (as compared to a completely distributed system) with the low communication overhead of direct peer-to-peer communication.

Although the evaluation of the initial implementation gave very promising results, there is much room for improvement, both in functionality and algorithmic performance. In particular more advanced strategies for pattern application in the server and support for queries involving concepts such as spacial proximity deserve future attention.

Acknowledgements

This work was supported by the Bayerische Forschungsstiftung (project TrackFrame, AZ-653-05) and the PRESENCCIA Integrated Project funded under the European Sixth Framework Program, Future and Emerging Technologies (FET) (contract no. 27731).

References

[1] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riss, C. Sandor, and M. Wagner. Design of a component-based augmented reality framework. In *Proceedings of the International Symposium on Augmented Reality (ISAR)*, Oct. 2001.

[2] G. Coulouris. Review report: The qosdream project. Technical report, Laboratory for Communication Engineering, University of Cambridge, 2002.

[3] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal*, 16 (2-4):97–166, 2001.

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

[5] F. Hohl, U. Kubach, A. Leonhardi, K. Rothermel, and M. Schwehm. *Next Century Challenges: Nexus - An Open Global Infrastructure for Spatial-Aware Applications*. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 249–255. Universität Stuttgart : Sonderforschungsbereich SFB 627 (Nexus: Umgebungsmodelle für mobile kontextbezogene Systeme), Seattle, WA, USA: not available, August 1999.

[6] A. MacWilliams, C. Sandor, M. Wagner, M. Bauer, G. Klinker, and B. Brügge. Herding sheep: Live system development for distributed augmented reality. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, Oct. 2003.

[7] H. Najafi, N. Navab, and G. Klinker. Automated initialization for marker-less tracking: A sensor fusion approach. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'04)*, Arlington, VA, USA, Nov. 2004.

[8] J. Newman, M. Wagner, M. Bauer, A. MacWilliams, T. Pintaric, D. Beyer, D. Pustka, F. Strasser, D. Schmalstieg, and G. Klinker. Ubiquitous tracking for augmented reality. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'04)*, Arlington, VA, USA, Nov. 2004.

[9] D. Pustka. Construction of data flow networks for augmented reality applications. In *Proc. Dritter Workshop Virtuelle und Erweiterte Realität der GI-Fachgruppe VR/AR*, Koblenz, Germany, September 2006.

[10] D. Pustka, M. Huber, M. Bauer, and G. Klinker. Spatial relationship patterns: Elements of reusable tracking and calibration systems. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'06)*, October 2006.

[11] D. Pustka, M. Huber, F. Echtler, and P. Keitler. UTQL: The Ubiquitous Tracking Query Language v1.0. Technical Report TUM-I0000, Institut für Informatik, Technische Universität München, 2007.

[12] 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.

[13] 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.