A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing

Christoph Endres, German Research Center for Artificial Intelligence, DFKI GmbH, Christoph.Endres@dfki.de
Andreas Butz, Munich University, Germany, butz@ifi.lmu.de
Asa MacWilliams, Technical University of Munich, Germany, macwilli@in.tum.de

To appear in "Mobile Information Systems Journal" 1(1), Jan–Mar 2005, IOS press, Amsterdam

Abstract

In this survey, we discuss 29 software infrastructures and frameworks which support the construction of distributed interactive systems. They range from small projects with one implemented prototype to large scale research efforts, and they come from the fields of Augmented Reality (AR), Intelligent Environments, and Distributed Mobile Systems. In their own way, they can all be used to implement various aspects of the ubiquitous computing vision as described by Mark Weiser [60].

This survey is meant as a starting point for new projects, in order to choose an existing infrastructure for reuse, or to get an overview before designing a new one. It tries to provide a systematic, relatively broad (and necessarily not very deep) overview, while pointing to relevant literature for in-depth study of the systems discussed.

1 Introduction

With the widespread availability of mobile computing devices, such as PDAs and smart phones as well as pervasive networks, such as GSM or WLAN, our attitude towards computing is changing rapidly. Computers have moved from our basements to our desks over the last three decades and they will move on from there into many parts of our daily environments. They will be connected to each other and equipped with extended sensing capabilities, making them reactive to their context of use. Today's mobile computing devices are only an intermediate stage in this evolution, but the vision of ubiquitous computing [60] has become realistic to the extent that big software companies and popular magazines start to talk about it, often coining their own terms, but basically describing similar ideas.

Research has picked up on this topic about a decade earlier and today many groups spend considerable amounts of effort and funding on the investigation of various aspects of ubiquitous computing. Often the first step towards research in this field is choosing or establishing a hardware and/or software infrastructure with which prototype scenarios can be developed and tested. These infrastructures or frameworks usually provide the basis for building ubiquitous computing applications as distributed interactive systems. They provide abstractions for networking, sensors or data, as well as formalisms and models for the specification of such systems.

Researchers new to ubiquitous computing will invariably have to catch up with a lot of this previous work in order to find out where others might already have solved aspects of their own research agenda. The purpose of this survey is to provide a starting point for this process and to enable researchers to quickly identify relevant work in their field which can then be studied in more detail.

2 Terminology and Criteria for the Survey

For the sake of this survey we have decided to subsume the various terms used in this field under three broad areas, which reflect fundamentally different approaches to building ubiquitous computing environments.

- **Augmented Reality (AR)** overlays a virtual layer to the physical environment and thereby makes computing power (mostly visually) appear in the environment although it is physically located elsewhere. This group also includes projects which are concerned with *mixed reality*.
- **Intelligent Environments (IE)** embed sensors, actuators and/or processors into the environment and thereby achieve behavior which was previously impossible. Parts of the ubiquitous computing power therefore reside in the actual objects of the environment, while others are still located on backend systems. This family also subsumes the terms *smart spaces*, *instrumented rooms*, as well as *embedded systems*.
- **Distributed Mobile Systems (DMS)** provide ubiquitous computing power by coordinating and integrating multiple mobile devices and distributing functionality across them. This family is closest to the original ubiquitous computing scenario and subsumes among others the term *context-aware computing*.

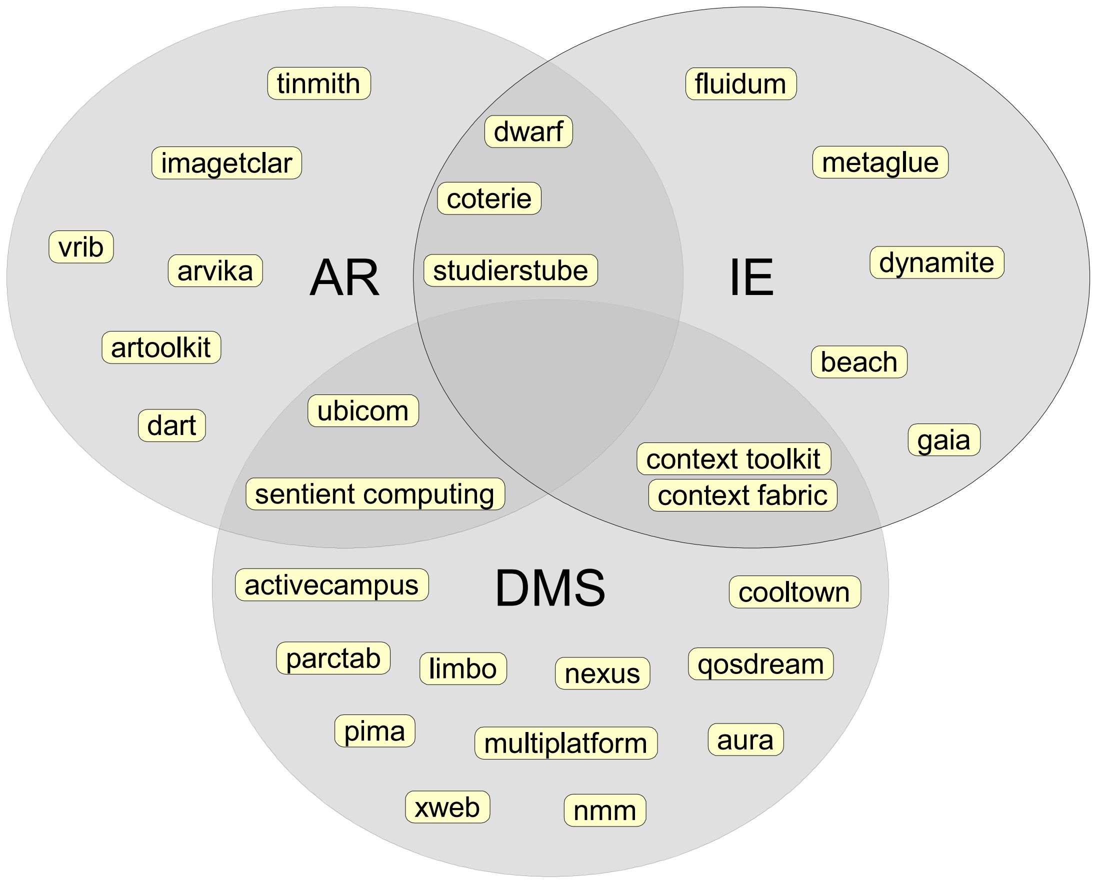

Figure 1: A thematic map of the systems discussed in this survey. Geometric proximity in the map signifies thematic relatedness.

We realize that this separation is a rather strong simplification. The 29 systems discussed in this survey span a wide spectrum, but we hope to provide a useful first classification within this vast landscape. Figure 1 presents a more subtle version of this classification, where borderline cases can be recognized as such and where close proximity also expresses relationships between systems. The placement of a system in the diagram roughly expresses its position between the three groups described above. The criteria we use to describe the systems are split into five major sections. They are:

1. **Type and background**
   Group/company: Research group or company from which the system originates.
   Time/manpower: Time period and manpower spent, as a measure for the effort that went into the system.
   Development focus: Focus for the development of the system, e.g., research or commercial.
   Research goals: Research goals for the group and the system, e.g., in terms of major scientific or design goals.
   Contact: Main contact person for the project, e.g., principal researcher and senior developer.
   Target environment: Specific application environment targeted by the system.

2. **System description**
   A very brief description of the system, partly in our own words, partly in terms used by the system's authors, where appropriate.
   Interesting aspects: Aspects, which distinguish this system particularly from others.

3. **Underlying technology**
   Language: Programming language, in which the system is implemented or in which applications are written.
   Network protocol: Protocols used for communication between system components and/or devices.
   Supported platforms: Hard- or software platforms on which the infrastructure runs.
   Scalability: Scalability of the infrastructure, e.g., number of users, devices, covered area.
   Underlying paradigm: Basic programming paradigm or conceptual model of the system.

4. **Components**
   Types: Types of system components.
   Granularity: Granularity at which the system allows to build these components.
   Description: Component descriptions and interface definitions, as used by compilers and run-time component management systems; e.g., IDL, XML, C++ class hierarchy
   Instantiation: of components at run time, e.g., command line, application, infrastructure
   Configuration: of components, e.g., component description, command line, config tool
   Communication/lookup: Organization of the communication between and lookup of components, e.g., central, decentral or hybrid.

5. **General information**
   Accessibility: Public accessibility of the system's source code, binaries, documentation, existence of discussion groups, etc.
   Level of abstraction: The level of abstraction at which a programmer has to specify applications.
   Modules and services: Predefined services or modules which are part of the infrastructure.
   Suitable for: Things for which the system is particularly well suitable.
   Key publications: These are often the first comprehensive publications about a system, but sometimes also the most recent and up-to-date papers. They are recommended as starting points for a detailed study of the system.

We consciously refrained from giving project URLs, since all these projects can easily be found by typing their names into any contemporary web search engine, while URLs might in some cases already be outdated at the time of publication of this article.

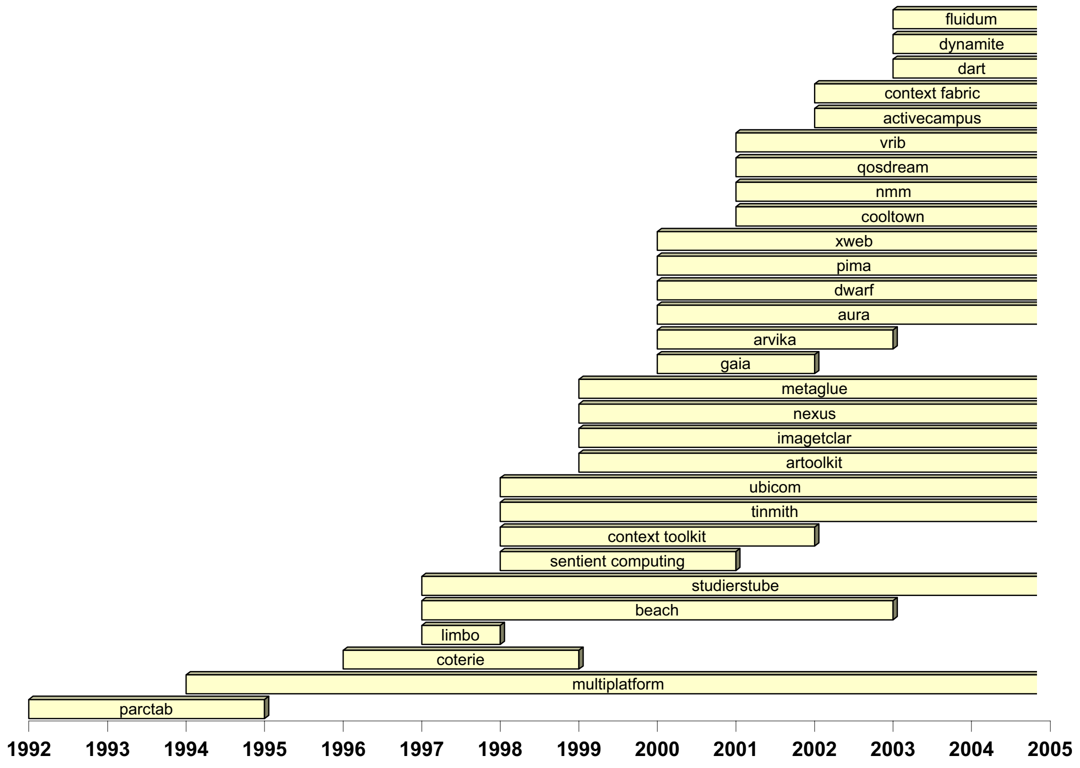

Figure 2: Time line of systems discussed in this survey

In order to provide a rough historical perspective, Figure 2 gives a chronological time line of the project durations. Open ended lines mean that the projects are still active. The following three main sections of the paper provide a list of these systems under the criteria stated above. The sections are split according to the three families of systems mentioned before, and within each section, systems are ordered alphabetically in order to avoid any apparent judgment of importance or relevance.

3 Augmented Reality Systems

This section contains infrastructures which are mainly used for implementing AR systems and prototypes. AR Toolkit is a publicly available library for marker recognition and camera-based tracking with a large user community. Arvika is a large research consortium who developed two infrastructures for their own purposes. Dart wraps trackers and other AR functionality into a Macromedia Director programming environment, while Coterie provided tracker abstractions and distributed graphical objects in a Modula-3 environment. Dwarf provides a collection of reusable software components for the quick assembly of AR applications and has recently been interfaced with Studierstube, a scene-graph-based AR infrastructure. ImageTclAR provides AR functionalities in aTcl/TK environment, while Tinmith does so in C++ with a focus on performance. UbiCom provides Quality of Service (QOS) mechanisms on mobile and wearable devices, and Vrib provides reusable software components for AR prototyping in C++ and Java.

3.1 AR Toolkit

1. **Type and background**
   Group/company: Human Interface Technology laboratory (HITlab), University of Washington, USA
   Time/manpower: First published in 1999, initial development by Hirokazu Kato and Mark Billinghurst; extensions, fixes and ports, partially by other groups until present.
   Development focus: Research.
   Research goals: To solve the recurring problem of position and orientation tracking in video-based and see-through AR to enable people to write their own (potentially collaborative) AR applications; To provide a toolbox for easy prototyping of AR applications, demonstrate possible uses of collaborative AR
   Contact: Hirokazu Kato, Mark Billinghurst
   Target environment: AR with Head-mounted Displays (HMD).

2. **System description**
   The AR Toolkit is basically a specialized computer vision library which analyzes the video stream of a camera, recognizes markers with their pose and ID, and provides marker position and orientation in a data structure suitable for OpenGL-based rendering.
   Interesting aspects: The AR Toolkit has a large user community, a robust recognition, and uses cheap printed markers.

3. **Underlying technology**
   Language: C, Java wrapper available.
   Network protocol: N/A
   Supported platforms: MS Windows, MacOS, Irix, Linux, Matlab, MS PocketPC.
   Scalability: Recognition accuracy decreases with the number of marker candidates in the image and with the number of possible marker IDs, so it doesn't scale well to thousands of possible markers.
   Underlying paradigm: OpenGL scene graph.

4. **Components**
   Types: Library for marker recognition, calibration routines.
   Granularity: Small (coordinates, matrices).
   Description: C API.
   Instantiation: By starting the recognition library.
   Configuration: Configuration files.
   Communication/lookup: N/A

5. **General information**
   Accessibility: Source code, binary distributions for all platforms, example programs, tutorials, manual.
   Level of abstraction: The programmer works with API calls, on a level similar to OpenGL programming.
   Modules and services: Recognition library, calibration procedures for different cameras.
   Suitable for: The AR Toolkit is particularly suitable for AR applications involving head-worn displays, because its pose recognition is weakest along the optical axis of the camera. This in turn is least noticeable if tracked objects are placed in the camera image. Consequently, it is less suitable for tracking scenarios, where absolute positions are needed and the scene is viewed from positions other than the tracking camera's own position.
   Key publications: The working principles and results about tracking accuracy are described in [28]. Further publications are listed on the AR Toolkit home page.

3.2 Arvika
Augmented Reality for Development, Production and Servicing

1. **Type and background**
   Group/company: The Arvika project was a joint academic/industrial project sponsored by the German federal ministry for education and research. It involved many partners from basic research, technology manufacturers and various application domains, e.g., automotive and machine maintenance.
   Time/manpower: From 2000 until 2003, with a large number of developers.
   Development focus: Research, with commercial applications as a main goal.
   Research goals: To bring together research and industry and promote the industrial acceptance of AR, particularly in development, production and service of complex technical products; with one architecture for high-end applications in product development, and one for mobile applications in production and service environments.
   Contact: Wolfgang Friedrich
   Target environment: AR.

2. **System description**
   The *stationary high-end solution* is intended for lab environments where high accuracy of tracking is mandatory. All components of the high-end solution run on one system, either on Windows 2000/NT, Linux, or SGI IRIX. On top of the operating system are the device integration interface *IDEAL*, the *AR browser*, and the tracking component. On top of these AR specific components is the application-specific software. IDEAL allows network-wide access to various tracking and interaction devices. A common interface, the *low level device interface (LLDI)*, provides an abstract model of different devices.

   The *mobile web-based solution* is based on documents; AR content is one type of media among others (HTML, PDF, or CAD data). The client-side mobile platform provides the typical AR functionality with localization, tracking, graphics, and integration of interaction devices. The client uses the Microsoft Internet Explorer 5 on Windows platforms as a thin client with a web browser as the interface. The *AR Browser* is a local ActiveX component for tracking and 3D visualization. When AR scenes are referenced in web pages, the AR Browser is started and displays the registered scene.

   The server side handles information management and enterprise integration. It is based on the Apache Tomcat application server. Each component is realized following the Java Beans specification. For autonomous mobile use, the server-side components can also be deployed on the client platform.
   Interesting aspects: Despite the use of different hardware and operating system platforms for the stationary and mobile systems, all software libraries concerning rendering, tracking and device interface can be reused from a single source tree. Thus, the *AR Browser* runs both as a standalone high-end application and as an ActiveX component. The architecture for the Arvika mobile system is one of the only systems to consider the integration of data from enterprise systems.

3. **Underlying technology**
   Language: C++, Java.
   Network protocol: Custom network interface for tracking *(IDEAL)*; *HTTP* for content between server and mobile client.
   Supported platforms: High-end system: Windows 2000/NT, Linux, SGI IRIX. Mobile system: Windows platforms. Server: Linux, Windows.
   Scalability: Few users; mobile system for industrial environments.
   Underlying paradigm: Scene graph; document based; client/server.

4. **Components**
   Types: Mobile system: *ActiveX components*; server: *Java Beans.*

Granularity: Large (mobile client has a single AR Browser component).
Description: ActiveX components and Java Beans; additional Low-Level Device Interface for tracker abstraction.
Instantiation: High-end system: by application; mobile system: by web browser.
Configuration: High-end system: By application; mobile system: by web server.
Communication/lookup: Centralized.

5. **General information**
Accessibility: Available to Arvika members.
Level of abstraction: The programmer works with a document-based approach.
Modules and services: *Client:* 3D scene graph rendering; video-based collaboration; data exchange between client systems; location-based information selection; tracking; video streaming; tracking and interaction device integration (IDEAL); 3D interaction; information caching. *Server:* context management; document model based on the structure of machines; database and file access; integration of enterprise systems as information sources; workflow modeling; collaboration management; user interface adaption; annotation.
Suitable for: Arvika is especially suitable for industrial AR applications.
Key publications: [4, 16]

3.3 Dart Designer's Augmented Reality Toolkit

1. **Type and background**
Group/company: AEL (Augmented Environments Laboratory), Georgia Institute of Technology, Atlanta, USA
Time/manpower: Since 2003, with 3 researchers.
Development focus: Research.
Research goals: To enable designers to rapidly develop and test their AR experiences in the deployment environment, with a focus on supporting early stages of design.
Contact: Blair MacIntyre
Target environment: AR.

2. **System description**
The Dart system is based on the Macromedia Director. It uses familiar Director paradigms of a score, sprites and behaviors to allow a user to create complex AR applications. Dart also provides low-level support for the management of trackers, sensors, and camera.
Interesting aspects: Dart is a multimedia programming environment built as a collection of extensions to the Macromedia Director.

3. **Underlying technology**
Language: LINGO (scripting language of Macromedia Director), C++.
Network protocol: Based on *VRPN* (Virtual Reality Peripheral Network) [56].
Supported platforms: Any Macromedia Director supporting platform.
Scalability: Not specified.
Underlying paradigm: Plugin.

4. **Components**
Types: Macromedia Director *extensions*.
Granularity: Large (plugins for Xtras, cast, and behavior).
Description: Macromedia Director internal interface.
Instantiation: By application.
Configuration: Macromedia Director internal interface.
Communication/lookup: Centralized (direct communication).

5. **General information**
Accessibility: Papers and mailing list are available. A download area on the web page

provides software, manuals, and documentation.
Level of abstraction: The programmer works with Macromedia Director and its scripting language LINGO.
Modules and services: Director Xtra for communication with hardware (cameras, marker tracker, hardware trackers, sensors, and distributed memory). Collection of director behavior patterns with drag and drop behaviors for controlling the functionality of the AR application.
Suitable for:
Dart is especially suitable for rapid prototyping of AR experiences that use see-through displays.
Key publications: [17]

3.4 Coterie The Columbia Object-oriented Testbed for Exploratory Research in Interactive Environments

1. **Type and background**
   Group/company: CGUI (Computer Graphics and User Interfaces) lab, Columbia University, New York, USA
   Time/manpower: From 1996 until about 1999, with typically 1-2 PhD students.
   Development focus: Research.
   Research goals: To provide a testbed for fast prototyping of distributed virtual environment and AR systems.
   Contact: Blair MacIntyre (research), Steven Feiner (lab director)
   Target environment: Shared AR environments with multiple users, devices, trackers, and see-through head-worn displays (HMD), sharing a common virtual layer.

2. **System description**
   Coterie supports the creation of virtual environments with multiple simultaneous users interacting with many heterogeneous displays and input devices. It is designed around a multi-threaded, modular, object-oriented programming model and supports transparent distributed communications via both client-server and replicated distributed objects. Applications developed on top of it include EMMIE, an environment manager providing interaction techniques for AR environments[6] and MARS (Mobile Augmented Reality Systems), the lab's series of mobile AR prototypes[14].
   Interesting aspects: Coterie worked efficiently on three operating systems and thus enabled the transparent cooperation between vastly different types of machines. EMMIE provided hybrid forms of interaction between the physical and the virtual worlds. MARS integrated several tracking technologies into Coterie and thus allowed the exploration of mechanisms for roaming between these technologies.

3. **Underlying technology**
   Language: Modula-3, Obliq, Repo3D.
   Network protocol: *IP*, *TCP*, *UDP*.
   Supported platforms: MS Windows, Linux, Solaris.
   Scalability: EMMIE was demonstrated with 3 users and 6 devices on $10m^2$ tracked area. MARS was demonstrated with 2 users on several city blocks of DGPS registered area.
   Underlying paradigm: Shared scene graph, 3D objects with locally modified appearances and behaviors. By building applications as groups of cooperating threads, a single programming model can be used for both single and multiprocess programs.

4. **Components**
   Types: 3D *graphical objects* in a shared scene graph with behaviors and local modifications, *tracker objects*.
   Granularity: Medium (graphical objects for primitives, groups, polygon sets, and images; with behaviors).

Description: By application.
Instantiation: Using scripts.
Configuration: Using scripts.
Communication/lookup: Hybrid.

5. **General information**
Accessibility: Research papers only.
Level of abstraction: The programmer works with graphical objects, behaviors, and appearances, which are defined in script files.
Modules and services: Coterie provided an AR core system with drivers for different tracking technologies (GPS, ultrasonic, infrared, IR beacons). EMMIE contained an environment manager, distributed over all client machines, providing basic mechanisms such as drag and drop or menus. For MARS there is an authoring tool for situated content.
Suitable for: Coterie is especially suitable for exploration and fast prototyping of collaborative AR and VR applications. It is probably not suitable for less graphical applications, since everything relies on a shared scene graph.
Key publications: The whole system has been described in [33] and the underlying distributed graphics library was presented in detail in [34].

3.5 Dwarf Distributed Wearable Augmented Reality Framework

1. **Type and background**
Group/company: Institut für Informatik, Technische Universität München, Germany
Time/manpower: Since 2000, with 6 active researchers and many undergraduate students.
Development focus: Research.
Research goals: To provide an extensible, decentralized software framework for research in distributed AR and its convergence with Context-Aware and Ubiquitous Computing.
Contact: Gudrun Klinker (research), Asa MacWilliams (architecture)
Target environment: distributed AR

2. **System description**
Dwarf consists of reusable software services for building distributed AR systems; supporting decentralized CORBA-based middleware; and a generic AR software architecture. Services are described using XML. They are loosely coupled and may run in separate processes or on different network hosts. Services are composed dynamically by the middleware at run time, forming a distributed data flow graph. The middleware consists of one *service manager* on each network host. The service managers support service discovery, allowing new services and devices to be integrated at run time. Applications are modeled as a services providing configuration and glue logic.
Interesting aspects: Dwarf is entirely decentralized. The framework includes several development tools, e.g., for service configuration, system monitoring and data flow configuration, and modeling of multimodal interaction.

3. **Underlying technology**
Language: C++, Java, Python.
Network protocol: *CORBA IIOP* for communication; *SLP* for lookup; local interprocess communication and shared memory.
Supported platforms: Linux, Mac OS, Windows, StrongARM Linux; uses CORBA (OmniORB, OpenORB) for portability.
Scalability: Current implementation scales to several users in several rooms.
Underlying paradigm: Distributed services.

4. **Components**
Types: The basic unit of a Dwarf system is a *service*.

Granularity: Large (example services are a scene graph-based 3D viewer, or an optical tracking service).
Description: Components described in XML or via CORBA description interface; interfaces described in CORBA IDL.
Instantiation: Services may be started manually from the command line, or automatically by the middleware when they are required by other services.
Configuration: Services may be configured using the command line; interactively using a monitoring and configuration tool; using dedicated configuration services; or automatically by the middleware in response to changes of other services.
Communication/lookup: Decentralized lookup; P2P communication.

5. **General information**
Accessibility: Open source; online documentation and tutorials on web site; developer mailing list.
Level of abstraction: The application developer works with reusable services; the service developer works with interfaces and APIs.
Modules and services: Services are available for several trackers; distributed tracker configuration; calibration; scene graph-based 3D rendering; audio input and output; video capture; various interaction devices; multimodal interaction; 3D modeling; and run-time system development.
Suitable for: Especially DWARF is especially suitable for prototyping of distributed AR systems.
Key publications: [2, 35, 36]

3.6 ImageTclAR

1. **Type and background**
Group/company: Media Entertainment Technology Laboratory, Michigan State University, USA
Time/manpower: Since 1999. The system is based on the *ImageTcl* development environment, which started development in 1994. Up to 10 developers, including related projects that use and extend the system.
Development focus: Research.
Research goals: To support different types of developers in building AR systems: novice developers with little knowledge of AR; application developers who wish to reuse components; and advanced AR system developers.
Contact: Charles Owen
Target environment: AR.

2. **System description**
The ImageTclAR system is based on ImageTcl, an image processing library for Tcl/Tk. It consists of several C++ components, and infrastructure to build glue logic in Tcl. The C++ components are configured and composed using Tcl. Application logic is written in Tcl as well. This hybrid approach allows novice developers to build applications quickly, and lets experienced developers build powerful components.
Interesting aspects: ImageTclAR includes several development tools: an interactive component creation utility to easily define new data types or components; a build utility; and a graph editor to generate Tcl script code defining the data flow between C++ components.

3. **Underlying technology**
Language: Tcl/Tk, C++.
Network protocol: N/A
Supported platforms: Windows; originally, Unix as well.
Scalability: N/A
Underlying paradigm: Scripted components.

4. **Components**
   Types: Compiled C++; Tcl scripts
   Granularity: Large (C++ components e.g., for tracking, calibration, display).
   Description: Using ImageTcl mechanisms; interfaces defined using interactive component creation utility.
   Instantiation: C++ components are instantiated via Tcl glue logic.
   Configuration: Using scripts (C++ components are parameterized via Tcl scripts).
   Communication/lookup: Centralized (via Tcl glue logic; single host).

5. **General information**
   Accessibility: Free for non-commercial use.
   Level of abstraction: The programmer works with reusable data flow components for applications and C++ code for component development.
   Modules and services: C++ components are available for various tracking systems; various types of calibration; VRML import and 3D rendering.
   Suitable for: IMAGETCLAR is especially suitable for rapid prototyping of AR user interfaces, and for conducting HCI user studies.
   Key publications: [43]

3.7 STUDIERSTUBE

1. **Type and background**
   Group/company: Technical Universities of Vienna and Graz, Austria
   Time/manpower: Since 1997, with approx. 5 active developers.
   Development focus: Research, with application partners especially in medical research.
   Research goals: To explore three-dimensional interaction and new media in a general work environment, where a variety of tasks are carried out simultaneously; to find 3D interaction metaphors as powerful as the desktop metaphor for 2D.
   Contact: Dieter Schmalstieg
   Target environment: Collaborative AR.

2. **System description**
   STUDIERSTUBE is based on *OpenInventor*, a powerful scene graph library. It takes advantage of the capabilities of the scene graph wherever possible. STUDIERSTUBE runs as a *workspace* executable. Several users, each with his own workspace, can collaborate, using a shared scene graph. The scene graph synchronization is handled by a custom reliable multicast protocol. Workspaces find each other using a central *session manager*. For tracking and user input, STUDIERSTUBE uses *OpenTracker*, an XML-configurable data flow framework that handles input from several tracking sources. For interaction, STUDIERSTUBE provides the *personal interaction panel*, a handheld 2D virtual menu, as well as a 3D window manager, allowing users to switch between applications.
   Interesting aspects: STUDIERSTUBE is one of the longest-standing AR frameworks. A very large number of applications have been built using it.

3. **Underlying technology**
   Language: C++, Inventor, Python.
   Network protocol: *DIV* (Distributed Open Inventor): custom reliable multicast protocol to synchronize scene graphs; *OpenTracker*: custom multicast protocol for tracking data.
   Supported platforms: Windows, Linux, Mac OS; uses *Coin* and ACE (Adaptive Computing Environment) libraries for portability.
   Scalability: Several simultaneous users, usually in single room.
   Underlying paradigm: Scene graph.

4. **Components**
   Types: Inventor scene graph *nodes*; OpenTracker data flow *nodes*.

Granularity: Medium (functionality encapsulated in OpenInventor nodes).
Description: *OpenInventor* API and node kit definitions; XML description in *OpenTracker*.
Instantiation: *OpenInventor*; loads shared libraries.
Configuration: Configuration files (using fields in Inventor `.iv` file).
Communication/lookup: Centralized (uses *Session Manager* as central server that several *Workspace* clients connect to).

5. **General information**
Accessibility: LGPL and GPL, source available, actively supported.
Level of abstraction: The programmer can develop entirely in OpenInventor, reusing predefined nodes, without writing C++ code, or implement his own Inventor nodes in C++.
Modules and services: Many reusable Inventor Node Kits for display and interaction, notably the Personal Interaction Panel.
Suitable for: STUDIERSTUBE is especially suitable for collaborative, immersive AR, especially manipulating virtual 3D objects.
Key publications: [47, 49]

3.8 TINMITH

1. **Type and background**
Group/company: University of South Australia, Australia
Time/manpower: Since 1998; major redesign in 2000, with an average of 1 developer.
Development focus: Research, several iterations built in cooperation with Australian Office of Defense.
Research goals: To develop a stable infrastructure for mixed reality applications, with high performance even on resource-constrained mobile platforms as the major design goal.
Contact: Wayne Piekarski
Target environment: Mobile AR.

2. **System description**
The TINMITH software architecture is based on data flow from sensors, through the application logic, to rendering. Data flows via callbacks between lightweight C++ objects, which reside in a custom-implemented, in-memory hierarchical *object store*. Object classes are defined using C++ headers. Additional macros are parsed by a specialized compiler to generate code for serialization and callbacks. The objects are organized in a hierarchical object store, which is modeled on the Unix file system. The object store provides serialization and deserialization features. Callbacks and listeners form a directed data flow graph which is independent of the object store hierarchy. TINMITH provides a complete library of components which can be joined together to write complex applications, e.g., outdoor AR modeling.
Interesting aspects: The entire system is optimized for performance and runs stably on low-powered hardware. As a simple remote debugging tool, the object store can export all data as an NFS server.

3. **Underlying technology**
Language: C++.
Network protocol: Custom *XML* over TCP/IP and binary over UDP protocols
Supported platforms: Linux, FreeBSD, Windows.
Scalability: Designed for small number of independent users, but roaming in wide area.
Underlying paradigm: Data flow; hierarchical object store

4. **Components**
Types: C++ *objects* with data flow, hierarchical addressing and serialization mechanisms.
Granularity: Varies (from very small like numeric values to medium like tracker; composition supported).

Description: C++ classes.
Instantiation: By application, or by serialization library.
Configuration: By application, and using features of hierarchical object store, e.g., symbolic links.
Communication/lookup: Centralized (local lookup via object store; P2P network communication, but without discovery).

5. **General information**
Accessibility: Not specified.
Level of abstraction: The programmer works with reusable C++ classes and libraries.
Modules and services: Application support (menus, dialogs); 2D/3D rendering (using a custom scene graph); 3D geometry and coordinate system transforms; low-level I/O, e.g., for various trackers.
Suitable for: TINMITH is especially suitable for mobile AR, especially on small systems.
Key publications: [46]

3.9 UBICOM Ubiquitous Communications

1. **Type and background**
Group/company: Delft University of Technology (TU Delft), Netherlands
Time/manpower: Since 1998, with approx. 18 full-time developers.
Development focus: Research.
Research goals: To specify and develop resource-constrained wearable systems for mobile multimedia communications, using a system approach with negotiated quality of service.
Contact: R. L. Lagendijk
Target environment: Mobile Multimedia, mobile AR.

2. **System description**
The architecture of all UBICOM systems is heavily influenced by quality of service and resource constraints. A QoS management mechanism called *Adaptive Resource Contracts (ARC)* is used to describe QoS requirements between components, evaluate tradeoff curves, and optimize according to application-defined criteria. ARC is decentralized, in that it assumes that no component has system-wide knowledge. UBICOM uses a client/server design, whereby a mobile AR client performs some, but not all, tracking and rendering locally. This allows an optimized tradeoff between latency, rendering quality and resource use. For example, the server performs polygon reduction of complex virtual objects, and generates bitmapped "imposters" of distant objects, which the mobile client can display quickly. As the user approaches a virtual object, the higher-quality representation is chosen. On the wearable system, the software components are distributed on a set of hardware modules. Each of them has a SA 1100 computing running Linux and function-specific code. The modules are *positioning*, *rendering and display*, *video and application*, *wireless connection*, and *interconnect*. Each module uses function-specific hardware components, e.g., a GPS receiver, or a camera.
Interesting aspects: UBICOM follows a system approach, building custom software on custom hardware. This gives the developers full control of the design space.

3. **Underlying technology**
Language: Not specified.
Network protocol: Custom-designed wireless network technology.
Supported platforms: Custom-built *SA 1100* computing units running StrongARM Linux.
Scalability: Designed to cover a campus area with several simultaneous users.
Underlying paradigm: Distributed components; distributed QoS management; client/server.

4. **Components**
Types: Software components on dedicated hardware modules.

Granularity: Medium (renderer, tracker).
Description: QoS operational characteristics.
Instantiation: Not specified.
Configuration: By application.
Communication/lookup: Decentralized configuration and communication; unspecified lookup.

5. **General information**
Accessibility: Not specified.
Level of abstraction: The programmer works with distributed components and QoS parameters.
Modules and services: Wireless communication, QoS management, tracking, rendering, model reduction, video communication.
Suitable for: UbiCom is especially suitable for distributed mobile multimedia and mobile AR.
Key publications: [31, 44]

3.10 Vrib / Vario

1. **Type and background**
Group/company: TU Ilmenau and DaimlerChrysler R&D, Germany
Time/manpower: Since 2001.
Development focus: Research, with commercial applicability as a goal.
Research goals: To develop a toolkit *(Virtual Reality Interaktionsbaukasten)* for the rapid combination of hardware and software components in virtual and mixed reality, with both hardware (reusable sensor components, electronics, wiring boards etc.) and a software architecture, called Vario.
Contact: Ralph Schoenfelder
Target environment: AR, VR.

2. **System description**
The Vario architecture is based on the ideas of modularization and data flow graphs. Hence, the basic reusable components form the nodes of a data flow graph, communicating over a network. The components are managed by a run-time infrastructure consisting of three main units. The *central system manager (CSM)* exists exactly once and is the central point of configuration and control. A run-time configuration user interface controls the CSM. On each host in the network, there is one *daemon*, which communicates with the CSM, and can start new processes in which components run. In each process, there is one *process manager*, which communicates with the daemon, and starts and configures components within a process. Communication resources are managed at the appropriate level (e.g., daemon sets up local inter-process communication).
Interesting aspects: To aid development of new components, a builder tool generates stubs for components from XML descriptions. Vrib is an ambitious project, aiming to improve both hardware and software development for AR.

3. **Underlying technology**
Language: C++, Java (infrastructure available in both languages).
Network protocol: Custom *XML TCP/IP protocol* for configuration and control; custom binary *UDP* protocol, described in *XML*, for data flow; local interprocess communication and shared memory.
Supported platforms: Not specified.Uses ACE (Adaptive Computing Environment) and QT (an application framework from Trolltech) for portability.
Scalability: Designed to scale to several users within one room.
Underlying paradigm: Distributed communicating components.

4. **Components**
   Types: *Nodes* in a data flow graph, which can be configured by the run-time infrastructure.
   Granularity: Medium (trackers, renderers).
   Description: Custom XML format.
   Instantiation: By run time infrastructure.
   Configuration: By application.
   Communication/lookup: Central lookup, decentralized P2P communication.

5. **General information**
   Accessibility: Not specified.
   Level of abstraction: The programmer works with components in data flow graph.
   Modules and services: Device access, tracking, rendering.
   Suitable for: VARIO is especially suitable for rapid prototyping of AR/MR setups.
   Key publications: [50]

4 Intelligent Environment Systems

This second group lists infrastructures which are mainly used for Intelligent Environments. BEACH provides information sharing and collaboration support in a SMALLTALK programming environment, DYNAMITE has a focus on interoperability of very different types of devices, the FLUID MANAGER provides device management and unified APIs to heterogeneous devices in order to enable fast prototyping of interaction techniques in Java, and GAIA is a distributed object system with its own scripting language. METAGLUE is a Java-based infrastructure for larger instrumented environments with multiple users.

4.1 BEACH
Basic Environment for Active Collaboration with Hypermedia

1. **Type and background**
   Group/company: AMBIENTE group, Fraunhofer IPSI, Germany
   Time/manpower: From mid 1997 until end of 2003, with a total of about 15 people.
   Development focus: Research.
   Research goals: AMBIENTE: To investigate human-centered technologies for workplace interaction environments; BEACH: To support synchronous collaboration in work environments, especially meetings.
   Contact: Norbert Streitz (research), Peter Tandler (architecture)
   Target environment: Instrumented Environments on room- and building- level.

2. **System description**
   The system has a grid architecture with four horizontal layers of abstraction (core layer, model layer, generic layer, task layer) and five vertical slices of basic concerns (interaction model, environment model, user-interface model, application model, data model). A third dimension is the degree of coupling between the components in the architecture. This approach offers flexibility needed for heterogenous devices and the inclusion of new or future devices. At the core of the architecture is a *shared object space* which supports the development of higher-level functionality.
   Interesting aspects: BEACH allows synchronous collaboration by building on shared states of objects and application.

3. **Underlying technology**
   Language: Smalltalk.
   Network protocol: *TCP/IP.*
   Supported platforms: Any platform supporting Smalltalk; mainly tested on windows and some early prototypes on Solaris machines. Currently some weak dependencies to native

windows applications that could easily be removed.
Scalability: One server per room handles a room with up to 15 people using up to 20–30 devices.
Underlying paradigm: The system builds on the concept of sharing state among the cooperating devices within a common working context. By using shared state, developers can easily model the relevant aspects of collaboration situations, abstracting from distribution issues.

4. **Components**
Types: *Services.*
Granularity: Large (application parts).
Description: Smalltalk interface.
Instantiation: By application on request to the framework.
Configuration: Components are dynamically assembled and register their configuration parameters to the system. The user can configure them over a configuration GUI.
Communication/lookup: Centralized. One generic server (shared object space) per room takes care of synchronizing replicated objects. Naming service is available in the system.

5. **General information**
Accessibility: On request (negotiable).
Level of abstraction: The programmer works with applications in the architecture's grid. It is relatively easy to get started; optimizing and higher level functionality is more difficult.
Modules and services: Gesture recognition, presentation adaptation, support for interaction modes, support of sensors, representation of external state, support for basic document types, tools for meetings and creative work, support for web servers.
Suitable for: The design and concept of Beach turned out to be useful, but the implementation is a prototype that was not intended for further extensions.
Key publications: [53, 54, 55]

4.2 DynaMITE
Dynamic Multimodal IT Ensembles

1. **Type and background**
Group/company: Fraunhofer IGD, Germany, in cooperation with Loewe and the European Media Laboratory
Time/manpower: From 10/2003 until 09/2006, with approximately 10 researchers involved.
Development focus: Research, Open Source development.
Research goals: To support the user through dynamic ad-hoc Device Ensembles. A main goal is the development of a suite of algorithms and protocols as a potential new standard.
Contact: Thomas Kirste
Target environment: Instrumented Environments, i.e. home environment.

2. **System description**
The system aims at enabling the spontaneous interaction of heterogenous devices and software components from different vendors in order to analyze the user's interaction, interpret his goals, and to realize them. The work is based on previous projects at Fraunhofer ITG, especially Embassi and Soda-Pop.
Interesting aspects: DynaMITE uses ontologies to enable automatic component collaboration. It has a dataflow-based approach using memoryless channels and a publish/subscribe mechanism with treshhold.

3. **Underlying technology**
Language: Current prototype in Java; the final product should not depend on a certain language.

Network protocol: Generic protocols over *TCP/IP*.
Supported platforms: Currently all Java platforms supported, but about to be extended.
Scalability: Untested, but probably high.
Underlying paradigm: Self-organized software framework.

4. **Components**
   Types: *Applications*.
   Granularity: Varies (medium–large).
   Description: Publish-subscribe mechanism.
   Instantiation: Component dependent.
   Configuration: Component dependent.
   Communication/lookup: Decentralized.

5. **General information**
   Accessibility: Large download section on the project's web page, including first prototypes and demonstrators.
   Level of abstraction: The programmer works with an agent topology consisting of channels and transducers.
   Modules and services: On goal execution level: devices, assistants, communication server; on goal identification level: dialog management, speech recognizer, virtual character, GUI, etc.
   Suitable for: DynaMITE is especially suitable for multi-modal assistant systems.
   Key publications: [23, 22]

4.3 Fluid Manager, project FLUIDUM FLexible User Interfaces for Distributed Ubiquitous Machinery

1. **Type and background**
   Group/company: FLUIDUM project, previously Saarland University, now Munich University, Germany
   Time/manpower: Since 2003, with one full time developer and an average of two students at a time.
   Development focus: Research.
   Research goals: *FLUIDUM:* To investigate new interaction techniques and interaction metaphors for instrumented environments, while attempting to formulate an interaction standard for those environments; Fluid Manager: To investigate general principles of component communication in instrumented environments. The focus is on device access and device-application communication; the solutions found here however could be reused for similar problems, e.g., handling of distributed services in Instrumented Environments.
   Contact: Andreas Butz (research), Christoph Endres (architecture)
   Target environment: Instrumented Environments of different scales.

2. **System description**
   The Fluid Manager provides the software infrastructure for instrumented environments in different scales. Its primary focus is the management of dynamic addition and removal of devices and services. The system provides a matchmaking mechanism between applications and devices, and a uniform programming interface to the devices for the application programmer, which is independent of hardware, network or driver details.
   Interesting aspects: The Fluid Manager has a device classification using property lists instead of taxonomy. It uses a hybrid component communication approach (both centralized and decentralized).

3. **Underlying technology**
   Language: Java.
   Network protocol: *RMI.*

Supported platforms: Any platform supporting Java (including RMI).
Scalability: Centralized version can easily handle room-level applications with up to 15 devices and 5 users. Decentralized version scales up better and is only limited by the underlying network infrastructure.
Underlying paradigm: Classification of devices through a list of their properties.

4. **Components**
Types: Java RMI *objects*.
Granularity: Medium (devices, device features, services).
Description: Meta-information using a common interface.
Instantiation: Command line.
Configuration: Configuration files (semi-automatically generated).
Communication/lookup: Hybrid (by default using a central server; P2P fallback mode).

5. **General information**
Accessibility: The source code and documentation is available from the developers on request in the spirit of the GPL. There is no public discussion group or forum so far.
Level of abstraction: The programmer works directly with devices over their APIs. He does not have to be concerned about hardware-, network- or driver-details.
Modules and services: The main modules are a central device manager server and its proxy, implementation of device- and device-property-objects, decentralized network functionality, and a basic lease-, priority- and security- handling.
Suitable for: The Fluid Manager is especially suitable for programming applications for instrumented environments spanning several devices. It is not intended for usage in AR systems.
Key publications: [12, 13]

4.4 Gaia
Active Spaces for Ubiquitous Computing

1. **Type and background**
Group/company: University of Illinois at Urbana-Champaign, USA
Time/manpower: From 2000 until 2002.
Development focus: Research.
Research goals: To investigate Active Spaces, where data and applications are associated with the user.
Contact: Roy Campbell
Target environment: Instrumented Environments, Active Spaces.

2. **System description**
Gaia supports mobile, user-centric active space applications. It is built as a *distributed object system*. The kernel consists of a component management core for component creation, destruction and upload, with currently seven services built on top of it (context service, context file system, component repository, event manager, presence service, space repository, and security service).
Interesting aspects: Gaia uses a MPCC (model, presentation, controller, coordinator) pattern, which is an extension of the MVC (model, view, controller). [5] pattern. Gaia has its own scripting language LuaOrb [8], based on Lua [27].

3. **Underlying technology**
Language: Various.
Network protocol: *CORBA IIOP*.
Supported platforms: Various.
Scalability: Not specified.
Underlying paradigm: Distributed Object System.

4. **Components**
   Types: *Services* (distributed objects).
   Granularity: Medium (components, nodes, component containers).
   Description: Through component repository service and component containers.
   Instantiation: In the *component management core.*
   Configuration: *Automatic configuration* service.
   Communication/lookup: Centralized (*Component repository* service).

5. **General information**
   Accessibility: Web page with a lot of documentation, but no software download.
   Level of abstraction: The application programmer needs to know GAIA's scripting language LuaOrb.
   Modules and services: See system description above.
   Suitable for: GAIA is especially suitable for use with portable applications that need to be dynamically partitioned and distributed to a variety of devices.
   Key publications: [48]

4.5 METAGLUE, MIT project OXYGEN

1. **Type and background**
   Group/company: Computer Science and Artificial Intelligence Laboratory, Massachusetts institute of technology (MIT), USA
   Time/manpower: Since 1999, currently 5 researchers and 15 students.
   Development focus: Research, with industrial (computer hardware and telecommunications) partners.
   Research goals: To develop interactive environments in which computers communicate with humans on the human's terms, not on the computer's.
   Contact: Howard Shrobe
   Target environment: Instrumented Environments, Mobile Computing.

2. **System description**
   The MIT project OXYGEN aims to provide mechanisms and technologies for the seamless integration of computing into our daily environments. It unites a large number of researchers (up to 250) with various interests, approaches and methodologies. It should be seen as a pool of ideas and concepts all contributing to the common goal of pervasiveness of computing, rather than a single software system. Nevertheless, there are system layers with a potential for reuse, such as the *pebbles* approach and the GOALS and METAGLUE frameworks. Technical information in this section refers to METAGLUE, as this is the relevant part for this survey.
   Interesting aspects: There was a very large scale research effort integrating specialists from various fields. This should make for high quality research in the detail solutions.

3. **Underlying technology**
   Language: Java (METAGLUE framework).
   Network protocol: *TCP/IP*-based proprietary replacement for *Java RMI.*
   Supported platforms: Windows, Linux (also on HP iPaq).
   Scalability: The goal is to provide very scalable mechanisms for arbitrary numbers of users and machines globally. Demonstrations have shown interaction with less than ten users and less than 30 machines in several rooms.
   Underlying paradigm: Small software components (pebbles); automatically combined to larger modules and dynamically reconfigurable.

4. **Components**
   Types: Large collection of component *modules*, such as speech understanding, data sources, vision and gesture recognition, developed in other parts of the OXYGEN project.

Granularity: Varies (from resources and devices to complex modules).
Description: Components (agents) are integrated on multiple levels within the Metaglue framework, providing a mix of formal and informal description.
Instantiation: Automatically by processes in the environment, using the GOALS planning component.
Configuration: Automatically by processes in the environment, using the GOALS planning component.
Communication/lookup: Apparently decentralized.

5. **General information**
Accessibility: The Metaglue framework can be downloaded as packages for various Linux distributions as well as MS windows. It has a documentation and a FAQ as well as a bug reporting facility.
Level of abstraction: The programmer works on the level of devices, resources, components, and agents.
Modules and services: Launcher, scheduler, etc.
Suitable for: The Metaglue framework seems particularly suitable for developing distributed information systems spreading over many devices and users, using agents as the basic underlying paradigm. It appears to have no particular strengths in graphics, so it might not be the first choice for distributed AR.
Key publications: The first paper [9] and the initial thesis [45] about Metaglue.

5 Distributed Mobile Systems

The third group finally gives an overview of infrastructures for distributed, context-aware mobile and wearable systems. The application scenarios are even more diverse than in the other two groups. ActiveCampus is an infrastructure for a campus-wide network of PDAs, Aura supports the modeling of user tasks and activities, the Context Fabric provides a strong formalism for modeling contexts, but is more scalable than the Context Toolkit, which had the same purpose. CoolTown is an industry effort to model localized information by attaching URLs to objects, Limbo provides a *tuple space* for the exchange of information between mobile devices, and Multiplatform facilitates the integration of a heterogeneous set of software components written in various target languages. While Nexus mainly defines the structure and interface languages between the components of location-aware systems, Nmm provides an efficient middleware for distributed multimedia applications under Linux. The ParcTab system was the first fully implemented ubiquitous computing system with proprietary handheld devices, an infrared network, mobile services and a large community of experimental users. Pima defines an application model for pervasive computing applications, QoSDream provides abstractions for location-aware applications in Java, Sentient Computing provides a programming framework around the Active Bat ultrasonic tracker, and XWeb allows uniform access to hierarchical *XML* data over an extension of the *HTTP* protocol.

5.1 ActiveCampus

1. **Type and background**
Group/company: Department of Computer Science and Engineering, University of California, San Diego, USA
Time/manpower: Since 2002.
Development focus: Research.
Research goals: To simultaneously support extensibility and tight integration in a context-aware infrastructure for campus-wide ubiquitous computing using PDAs. On the one hand, components must be tightly integrated to present a convincing user experience; on the other hand, they should be only loosely coupled, to allow extensibility.

Contact: William Griswold
Target environment: Context-aware Ubiquitous Computing.

2. **System description**
ActiveCampus uses a central server for all components except data acquisition and user interaction, in order to ease administration and to minimize requirements placed on mobile devices. Additionally, centralization provides greater freedom to organize (server-side) components according to extensibility concerns. Sensors and display devices communicate with the server using SOAP. The server is a web server running PHP, backed by an SQL database.

   The architecture contains five layers. At the top is the *device* layer, where mobile devices connect to the ActiveCampus server. Next is the *environment proxy* which abstracts raw sensor data. A *situation modeling* layer aggregates context from several sensors, whereas an *entity modeling* layer refines context over time. At the bottom is the *data* layer, which handles persistent storage.

   Data regarding entities (users, buildings etc.) is stored in a *normal form* in the database. For example, each user has a unique numeric identifier. All other information (name, picture) are associated with that identifier. Similarly, all position information is represented in a normal Cartesian form.

   Services are decoupled from another using *introspection*: each service has a method which returns whether the service may be invoked upon a certain entity. Thus, for example, a buddy service can easily be integrated with an e-mail service, so that users can send mail to buddies they were chatting with.
Interesting aspects: The ActiveCampus system has been experimentally deployed in a large environment.

3. **Underlying technology**
Language: C++.
Network protocol: *HTTP*, *SOAP*.
Supported platforms: Windows CE.
Scalability: Campus-wide, with 700 PDA users.
Underlying paradigm: Client/server.

4. **Components**
Types: *Services*, which are functionalities that can be invoked on real-world *entities*.
Granularity: Medium (example services are a buddy service or an e-mail service).
Description: Using an introspection interface.
Instantiation: Not specified.
Configuration: Not specified.
Communication/lookup: Centralized.

5. **General information**
Accessibility: Not specified.
Level of abstraction: The programmer works with services and entities.
Modules and services: PDA display, location service, buddy service, messaging service.
Suitable for: ActiveCampus is especially suitable for wide-area ubiquitous computing applications.
Key publications: [20]

5.2 Aura
Distraction-free Ubiquitous Computing

1. **Type and background**
Group/company: Carnegie Mellon University, USA, and industrial partners
Time/manpower: Since 2000, with a total manpower of approximately 30 person-years so

far.
Development focus: Research, with transfer to commercial applications.
Research goals: To provide the user with a digital "halo" of computing and information while trying to satisfy two competing goals: to maximize the use of available resources while minimizing the distraction of the user.
Contact: Mahadev Satyanarayanan (research), João Sousa (software architecture)
Target environment: Ubiquitous Computing.

2. **System description**
The core part of the system is a *task manager* called *prism*, which tries to minimize the distraction of the user in the following four cases: the user moves to another environment; the environment changes; the task changes; and the context changes. The prism has the following components available and communicates with them: One or more context observer, with possibly varying degrees of sophistication, an environment manager as gateway to environment and file access, and several service suppliers. Furthermore, the prism can also communicate with other prisms to allow the seamless relocation of the user and its tasks to another environment.
Interesting aspects: AURA uses a task-centered approach with platform-independent description and migration of the tasks.

3. **Underlying technology**
Language: Various (Java, C/C++, Lisp, etc.).
Network protocol: *TCP/IP*
Supported platforms: Linux, Windows.
Scalability: Currently evaluated.
Underlying paradigm: Decomposing tasks as coalition of services.

4. **Components**
Types: *Supplier* (native application wrappers) and *infrastructure* components.
Granularity: Medium (tasks, services, suppliers).
Description: XML-based markup format.
Instantiation: The suppliers are instantiated by the application, the infrastructure components (task manager *prism*, environment manager, context observer) by the AURA installation.
Configuration: Suppliers are configured dynamically, using AURA's infrastructure components.
Communication/lookup: Explicit implementation of connectors; suppliers are coordinated by AURA's infrastructure components.

5. **General information**
Accessibility: Consortium membership available.
Level of abstraction: The programmer works with the XML-based AURA protocols.
Modules and services: Infrastructure components (task manager "prism", environment manager, and context observer); various suppliers that act as adapters of interactive (desktop-like) applications to AURA. Currently implemented suppliers include wrappers for: web browser (Internet Explorer), office tools (Word, Excel, PowerPoint, GNU Emacs), media player (Windows Media Player, Xanim), speech recognizer (Sphinx), speech synthesizer (Festival), and a web-based translation service (BabelFish).
Suitable for: AURA is especially suitable for suspend/resume of user tasks as a logical unit, both on a single machine as well as across different locations.
Key publications: [51, 52]

5.3 CONTEXT FABRIC

1. **Type and background**
Group/company: Group for User Interface Research, UC Berkeley, USA

Time/manpower: Since 2002, currently 12 registered developers at sourceforge.net.
Development focus: Research.
Research goals: To provide an infrastructure for building context aware applications with mechanisms for privacy and security.
Contact: Jason I. Hong, James A. Landay
Target environment: Very large scale Instrumented Worlds (Instrumented Environments).

2. **System description**
   The Context Fabric is an alternative approach to modeling context from the research lab which currently hosts (but didn't originally develop) the context toolkit. It takes a much more scalable approach by using P2P networking right from the start.
   Interesting aspects: As with the Context Toolkit, this is a very strong formalization of the term context which imposes a clear structure and encourages a systematic approach to context awareness. On top of this it provides a scalable networking approach.

3. **Underlying technology**
   Language: Java.
   Network protocol: *HTTP.*
   Supported platforms: Handhelds, wearables, custom built embedded systems.
   Scalability: Apparently high, no absolute data available.
   Underlying paradigm: Context aware applications are described in terms of their context sources and sinks which feed into and are fed out of InfoSpaces. The InfoSpace supports basic tuple space operations, including query and subscription. It differentiates between permanently stored tuples and temporary tuples (for sensor readings). A *multi-hop communication manager* stores and forwards messages between tuple spaces. A *context rule engine* processes context rules based on the data in the tuple space and generates new tuples automatically.

4. **Components**
   Types: Context *sources* and *sinks*, information (tuple) *spaces*, context *rule sets.*
   Granularity: Small (one tuple per sensor reading).
   Description: Classes defined within the Context Fabric, with uniform interfaces.
   Instantiation: By application.
   Configuration: By application.
   Communication/lookup: Decentralized (multi-hop; P2P communication).

5. **General information**
   Accessibility: Java code can be downloaded from sourceforge.net; there is also a discussion forum and a bug tracker.
   Level of abstraction: The programmer works with context tuples, but the level of abstraction is really determined by the programmer himself, and can range from raw sensor data to high level descriptions.
   Modules and services: Servers for managing info spaces, subscriptions, operators etc.
   Suitable for: The Context Fabric seems especially suitable for implementing large scale context dependent applications in distributed environments.
   Key publications: [26]; documentation and source code are hosted at sourceforge.net.

5.4 Context Toolkit

1. **Type and background**
   Group/company: Previously Georgia Institute of Technology, now UC Berkeley, USA
   Time/manpower: From 1998 until 2002, mostly 3 researchers, now hosted at sourceforge.net.
   Development focus: Research.
   Research goals: To provide a formalism and framework for building context aware applications.

Contact: Daniel Salber, Anind K. Dey, Gregory D. Abowd
Target environment: Instrumented Environments containing multiple sensors.

2. **System description**
The Context Toolkit was developed as the implementation of a formalism for describing context in its various levels of abstractions. It is a textbook example for this kind of formalization. Context can be data from single context widgets (basically sensor drivers), but also context from several widgets aggregated by context aggregators, and finally translated by context interpreters (such as lookup tables). The toolkit also provides a lookup and subscription mechanism for applications to use this context.
Interesting aspects: Very strong formalization of the term context which imposes a clear structure and encourages a systematic approach to context awareness.

3. **Underlying technology**
Language: Java, but components exist in C++, Frontier, Visual Basic and Python. Interface clients exist for Flash and .NET.
Network protocol: *HTTP.*
Supported platforms: Handhelds, wearables, custom built embedded systems.
Scalability: Limited by centralized architecture.
Underlying paradigm: Context widgets (sensors) which can be aggregated (sensor fusion and composition) and interpreted.

4. **Components**
Types: Context *widgets*, *aggregators*, *interpreters*, *services* and *discoverers.*
Granularity: Varies.
Description: Java class hierarchy.
Instantiation: By application. **Configuration:** N/A.
Communication/lookup: Centralized (yellow and white page services).

5. **General information**
Accessibility: Java code can be downloaded from sourceforge.net; there is also a discussion forum and a bug tracker.
Level of abstraction: The level of abstraction is determined by the programmer himself; it can range from raw sensor data to high level descriptions.
Modules and services: Central lookup and subscription mechanism.
Suitable for: As the name already suggests, the Context Toolkit is especially suitable for implementing various types of context dependent applications in distributed environments. It doesn't feature any special support for AR or graphical applications.
Key publications: The whole system is nicely described and put into perspective in a comprehensive journal article [11].

5.5 CoolTown

1. **Type and background**
Group/company: Internet and Mobile Systems Laboratory, Hewlett Packard, Palo Alto, USA
Time/manpower: First publication in 2001, web site currently being updated.
Development focus: Research, with commercial background.
Research goals: To investigate the convergence of web technology, wireless networks, and portable devices.
Contact: Tim Kindberg
Target environment: Mobile Computing.

2. **System description**
CoolTown is an infrastructure for context-aware applications. Real world objects (people, places, devices) are represented with a web page. Those web pages automatically update

themselves when new information about the real world entity they represent becomes available.
Interesting aspects: CoolTown uses web servers for representation of real world entities, and sensing mechanisms (bar code reader, infrared, etc.) for obtaining URLs from real world objects and access their web representation.

3. **Underlying technology**
Language: Various (potentially any language supporting networking).
Network protocol: *HTTP*.
Supported platforms: Any network-enabled platform.
Scalability: Not specified, but probably high.
Underlying paradigm: Any real world object can be represented as a web page.

4. **Components**
Types: *Web server*.
Granularity: Large (applications).
Description: Access to components via HTTP protocol.
Instantiation: Usually command line.
Configuration: Implementation dependent.
Communication/lookup: Centralized (various servers).

5. **General information**
Accessibility: Not specified.
Level of abstraction: The programmer works with communication between real world entities via HTTP protocol.
Modules and services: Web servers and specialized hardware.
Suitable for: CoolTown is especially suitable for applications aimed at providing information and services to users.
Key publications: [7, 29]

5.6 Limbo

1. **Type and background**
Group/company: Distributed Multimedia Research Group, Lancaster University, UK
Time/manpower: In 1997.
Development focus: Research.
Research goals: To provide better support for adaptive mobile applications by using the tuple space paradigm [19] for communication.
Contact: Nigel Davies
Target environment: Ubiquitous Computing, Mobile Computing.

2. **System description**
The platform is based on the Linda tuple space model [3] and built on top of the MOST platform. It includes a number of significant extensions to Linda, mainly to address the specific requirements of mobile environments. Key extensions are: multiple tuple spaces, explicit tuple-type hierarchy with support for dynamic sub-typing, explicit QoS attributes in tuples, and a number of system agents that provide services for QoS monitoring, creation of new tuple spaces, and propagation of tuples between tuple spaces.
Interesting aspects: Limbo is the first mobile application infrastructure using tuple spaces with some interesting extensions of the original tuple space concept.

3. **Underlying technology**
Language: Not specified. (built on top of Lancaster University's MOST platform).
Network protocol: *QEX* [15] for ANSA applications; *TCP/IP* for Windows/NT applications.
Supported platforms: All MOST platforms and also Windows/NT via gateway.

Scalability: In the centralized version depending on the amount of tuple spaces in use and the synchronization overhead between them. The decentralized version is as scalable as the underlying network permits (i.e. usually very good).
Underlying paradigm: Multiple tuple spaces.

4. **Components**
Types: Various tuple space *clients*, e.g., ANSA *applications.*
Granularity: Varies.
Description: ANSAware IDL.
Instantiation: By application.
Configuration: N/A
Communication/lookup: The first prototype has a centralized tuple space with the option to create more, communicating tuple spaces. A second, decentralized version is mentioned in [10] as "currently developed", but no later reference could be found.

5. **General information**
Accessibility: Research papers only.
Level of abstraction: The programmer must be familiar with the QEX protocol [15].
Modules and services: Tuple space creation agents, bridging agents, and QoS monitoring agents for connectivity, power, and cost.
Suitable for: LIMBO is especially suitable for asynchronous communication in mobile applications.
Key publications: [10]

5.7 MULTIPLATFORM
Multiple Language / Target Integration Platform for Modules

1. **Type and background**
Group/company: Projects Verbmobil, Smartkom and COMIC, German Research Center for Artificial Intelligence (DFKI GmbH), Germany
Time/manpower: From 1994 until 2005, with an average of 2 active full time developers.
Development focus: Research, Open Source development.
Research goals: *VERBMOBIL:* To develop a speaker-independent, bidirectional speech-to-speech translation system; *SMARTKOM:* To develop a mixed-initiative dialog system; *COMIC:* To build a multi-modal, mixed-initiative dialog system; MULTIPLATFORM: To provide a general framework for building integrated natural language and multimodal dialog schemes for these projects.
Contact: Wolfgang Wahlster (research), Gerd Herzog (software architecture)
Target environment: Distributed Mobile Systems.

2. **System description**
MULTIPLATFORM is a general framework for building integrated natural-language and multimodal dialog systems. The approach relies on a distributed component model. It includes support for compiling and installing modules, means to start, control and stop the modules, a communication framework, support for different programming languages, an XML based interface framework, and various debugging tools.
Interesting aspects: MULTIPLATFORM is event-triggered, and has no central control mechanism.

3. **Underlying technology**
Language: Various (C, C++, Java, Perl, Prolog, Lisp).
Network protocol: *PCA* [30] over *PVM* [18]
Supported platforms: Windows, GNU/Linux, Solaris.
Scalability: Tested with up to 70 modules.
Underlying paradigm: Publish/subscribe component architecture.

4. **Components**
   Types: *Modules.*
   Granularity: Varies (from simple services to full applications).
   Description: Multimodal Markup Language M3L; *module functionality* service.
   Instantiation: Startup via script, later monitoring and restart if necessary through testbed application.
   Configuration: Configuration files.
   Communication/lookup: Centralized (event-triggered).

5. **General information**
   Accessibility: Documented open source project, publicly available at sourceforge.net, but marked as dormant. Developers mailing list with archive. There have been workshops for interested customers.
   Level of abstraction: The programmer works with system modules. The system has a relatively high learning curve. All documentation for the experienced programmer is available, but distributed over a lot of heterogenous documents.
   Modules and services: The middleware provides module manager, testbed manager, testbed GUI, logger, replay mechanism, and data viewer. Application modules used with the system include various recognizer, analyzer, generator, synthesizer, modeler, and services.
   Suitable for: MULTIPLATFORM is especially suitable for dialog systems.
   Key publications: [24, 25], also parts of [57, 58]

5.8 NEXUS Platform

1. **Type and background**
   Group/company: NEXUS SFB, University of Stuttgart, Germany
   Time/manpower: From 1999 until 2005, started with 5 researchers, now part of a bigger special research program.
   Development focus: Research.
   Research goals: To provide a structure and a software framework for large scale location dependent services
   Contact: Kurt Rothermel (research), Daniela Nicklas (architecture)
   Target environment: Large scale Location-aware Systems.

2. **System description**
   The NEXUS platform provides a structure for location based services (LBS) similar to the one of the world wide web. NEXUS servers contain localized information. NEXUS clients query information knowing their own location. Queries are modified in a federation layer and sub-queries are forwarded to the appropriate servers, and sub-results are combined again to form the overall result. Area Service Registers keep track of available localized information.
   Interesting aspects: By taking an approach similar to the structure of the WWW, the conceptual model will be intuitive for many developers, and scalability potentially corresponds to that of the WWW.

3. **Underlying technology**
   Language: Various: The NEXUS platform partially uses existing software, such as data base management systems and just specifies languages for querying and exchanging information as a dialect of XML, as well as a service interface for servers.
   Network protocol: *TCP/IP.*
   Supported platforms: Various.
   Scalability: Conceivably large, no quantitative information available.
   Underlying paradigm: Analogous to the WWW.

4. **Components**
   Types: Spatial Model *servers*, client *applications*, federation *nodes.*

Granularity: Varies (depending on application).
Description: Defined service API.
Instantiation: Not specified.
Configuration: Configuration files
Communication/lookup: Centralized (client-server model, lookup similar to DNS).

5. **General information**
Accessibility: Research papers about the architecture.
Level of abstraction: The programmer works on a very high abstraction level.
Modules and services: Lookup through area service registers.
Suitable for: NEXUS is especially well suitable for very large scale LBS at different degrees of localization.
Key publications: A good overview is provided in [41].

5.9 NMM Network-integrated Multimedia Middleware

1. **Type and background**
Group/company: Computer Graphics Lab, Saarland University, Germany
Time/manpower: Since beginning of 2001, with 1-2 full time developers and 4-5 students.
Development focus: Research, Open Source development.
Research goals: To design and develop multimedia middleware for (mainly) Linux, which considers the network as an integral part and enables the intelligent use of devices distributed across a network. On the kernel side this means building an open, integrating architecture which can include heterogenous systems. On the service side it means investigating principles of session sharing and hand-over.
Contact: Marco Lohse
Target environment: Multimedia, i.e. the framework can be used as enabling technology for traditional applications, but also for Ubiquitous Computing and Mobile Computing.

2. **System description**
Within NMM, all hardware devices and software components are represented by so called *nodes*. A *node* has properties that include its input and output ports. The system distinguishes between six different types of nodes: *source*, *sink*, *filter*, *converter*, *multiplexer*, and *demultiplexer*. The NMM architecture uses a uniform message system for all communication. There are two types of messages: *multimedia data* and *events*. Object-oriented interfaces allow control of objects by simply invoking methods. External listener objects can register to be notified when certain events occur at a node. At run time, supported events and interfaces can be queried by the application.
Interesting aspects: NMM has a meta-architecture which integrates heterogenous systems independent of underlying technology and thus enables new forms of middleware-services.

3. **Underlying technology**
Language: Mainly, but not limited to C++, Some proof of concept implementations in Java exist.
Network protocol: Flexible. Modules supporting various protocols can be plugged.
Supported platforms: Any platform running Linux. (PC, PDA, Set-top boxes, etc.)
Scalability: Theoretically unlimited in terms of amount of users and devices. Covered area limited only by network limitations.
Underlying paradigm: Flow graph.

4. **Components**
Types: *Nodes.*
Granularity: Small (as small as possible).

Description: Various kinds of module description (XML, API,...); generic IDL for interfaces, similar to CORBA.
Instantiation: Automatic detection of new hardware and registration in network-wide registry.
Configuration: Devices offer preferred configuration, but can adapt to application requirements.
Communication/lookup: Decentralized (using a P2P network).

5. **General information**
Accessibility: Well documented open source project with mailing lists, discussion group and WiKi.
Level of abstraction: The programmer works on a high level of abstraction: Applications are easily configurable using *clic*, a tool to build new applications simply by describing them in a text file.
Modules and services: Core modules include the kernel and approximately 70-80 plugins for multimedia input, output, and communication. Core services are network-wide registry and resource management. There is also support for session sharing, session hand-over and multimedia format negotiations available.
Suitable for: NMM is especially suitable for everything that can be described in a flow graph. Although the system is at the moment mainly used for multimedia, it might as well be used for web services, data processing, or any other data flow application. It is not intended for database-like tasks.
Key publications: [32, 37, 38]

5.10 Xerox PARCTAB

1. **Type and background**
Group/company: Xerox PARC (today: PARC, Palo Alto Research Center), USA
Time/manpower: From 1992 until 1995, with approx. 8 researchers plus many students.
Development focus: Research, and in-house use for evaluation.
Research goals: To provide a highly mobile device networked by a system of localized (Infrared) access points for the exploration of mobile and location aware applications within a closed user community.
Contact: Roy Want and Bill Schilit
Target environment: Early form of Ubiquitous Computing in a specially prepared environment (i.e. Instrumented Environments).

2. **System description**
Quoted from [59] for the lack of better words: "The PARCTAB system integrates a palm-sized mobile computer into an office network. This project serves as a preliminary testbed for Ubiquitous Computing, a philosophy originating at Xerox PARC that aims to enrich our computing environment by emphasizing context sensitivity, casual interaction and the spatial arrangement of computers."
Interesting aspects: The PARCTAB system was historically the first distributed info system with the declared goal of creating a ubiquitous computing environment. It included specifically designed mobile and networking hardware as well as a comprehensive software infrastructure and (given its limited range) it provided services which could still compete with current wireless mobile clients.

3. **Underlying technology**
Language: Modula-3, Tcl/Tk, MacTabbit.
Network protocol: Proprietary protocol over the IR link; *Sun RPC* between Unix machines.
Supported platforms: Sun Unix servers with SunOS 4, custom built hardware around 87C524 Microcontroller.

Scalability: Evaluated with 41 users and 50 IR cells in the PARC institute.
Underlying paradigm: A software agent responsible for each device, who follows it when it is roaming and provides services to it.

4. **Components**
Types: Mobile *units* with mostly display capabilities, *applications* on backend servers, managed by tab *agents*.
Granularity: Large (applications).
Description: New applications can be implemented using libraries for the Tab's GUI etc.
Instantiation: By run time infrastructure.
Configuration: Configuration files; network interface with command line.
Communication/lookup: Centralized.

5. **General information**
Accessibility: Hard-and software were accessible to other researchers at the time.
Level of abstraction: The programmer could write applications in Modula-3, just using the Tab's widget library, but also in Tcl/TK.
Modules and services: One tab agent per tab, IR gateways, tab shell to start applications.
Suitable for: PARCTAB was designed and hence is particularly well suitable for the exploration of mobile location-aware services within a closed community of users.
Key publications: [59] describes the Ubiquitous Computing philosophy, the PARCTAB system, user-interface issues for small devices, and the authors' experience developing and testing a variety of mobile applications."

5.11 PIMA Platform-Independent Model for Applications

1. **Type and background**
Group/company: IBM T.J. Watson Research Center, USA
Time/manpower: Since 2000.
Development focus: Research.
Research goals: To develop design time, load time, and run time infrastructure support for pervasive computing, with devices as portals into an application/data space; applications as a means by which a user performs a task; and the computing environment as the user's information-enhanced physical surroundings.
Contact: Guruduth Banavar
Target environment: Pervasive Computing.

2. **System description**
In [1], the authors propose a system model for pervasive computing (which was not implemented at that time). This model is described by design time, load time and run time. At design time, applications are modeled as abstract tasks, which require certain services. The developer is encouraged to focus on a particular task. Interaction elements are specified abstractly, capturing user intent rather than a specific representation. Required services are described in an abstract service description language. At load time, devices are specified by their capabilities, and the infrastructure matches this with applications' requirements. The system discovers and composes the system in order to perform the desired tasks. This involves dynamic discovery, pruning of hostable functions and adaptation of the presentation. At run time, the system handles redistribution, disconnection and failure recovery, when parts of the system change or fail.
Interesting aspects: The model proposed for PIMA is very broad in its application; implementing these concepts will probably require substantial future research.

3. **Underlying technology**
Language: Not specified.

Network protocol: Not specified.
Supported platforms: Not specified.
Scalability: Not specified.
Underlying paradigm: Service composition.

4. **Components**
Types: *Services*, *devices*.
Granularity: Medium (task, device).
Description: Abstract description.
Instantiation: By run time infrastructure.
Configuration: By application.
Communication/lookup: Not specified.

5. **General information**
Accessibility: Not specified.
Level of abstraction: The programmer works with tasks and service descriptions.
Modules and services: None at the time of the paper.
Suitable for: Pima is especially suitable for pervasive computing with a wide range of interaction devices.
Key publications: [1]

5.12 QoSDream Quality of Service for Dynamically Reconfigurable and Adaptive Multimedia

1. **Type and background**
Group/company: Laboratory for Communications Engineering, Cambridge University, UK
Time/manpower: Since 2001.
Development focus: Research.
Research goals: To develop a middleware framework for the construction and management of context-aware multimedia applications, with a uniform high-level data flow model; application-defined quality of service constraints; a sensor-independent spatial model of the world; event filtering and abstraction; and persistence.
Contact: George Coulouris
Target environment: Location-aware Systems.

2. **System description**
QoSDream includes a *location service*, which processes sensor data and handles location information; an *event service*, based on the CORBA notification service, for passing events to applications and between applications; a *distributed object database* for storage of persistent and static information; and a *distributed multimedia service* for management of data flow between components. Recently, the open source *Framework for Location Aware Modeling (*FLAME*)* has been developed, which extends the location related aspects of QoSDream, but without the multimedia components.

The core of QoSDream is the location service. *Federators*, or *technology adapters* in FLAME, process incoming sensor data. Federators are available for different sensing technologies, such as the *Active Bat* system, or for active badges. The federators generate events which are sent to the *spatial relation manager*. The spatial relation manager organizes these events into *regions*, using information from technology-specific *location modules*. For example, a person wearing an active badge has an associated region depending on the accuracy of the sensing technology, and a "visibility region" which is the area in front of the person.

Event adapters detect overlaps between the regions. They can detect, for example, when the visibility region of a person overlaps with that of a computer screen. The event adapters send

overlap events to the applications, which can react accordingly. Static location information, such as the location of rooms and walls, are stored in an object-oriented database.
Interesting aspects: Although originally intended to deal with general multimedia applications, both QoSDream and FLAME now focus on the management of spatial information.

3. **Underlying technology**
Language: Java, CORBA interface.
Network protocol: *CORBA IIOP* notification service
Supported platforms: Not specified.()
Scalability: Tested with the *Active Bat* tracking system; scales to medium-sized buildings.
Underlying paradigm: Client/server; publish/subscribe.

4. **Components**
Types: *Federators* (*technology adapters* in FLAME) for processing incoming sensor data; *location modules* to model sensor regions; *event adapters* to discover high-level events; applications, which use the data generated by the other components
Granularity: Large (tracking technologies, high-level event types, applications).
Description: Java class hierarchy.
Instantiation: By application.
Configuration: By application.
Communication/lookup: Centralized.

5. **General information**
Accessibility: FLAME is available as open source.
Level of abstraction: The programmer works with high-level spatial events.
Modules and services: Location service, event service, database service.
Suitable for: QoSDream is especially suitable for large-scale location-aware systems.
Key publications: [39]

5.13 Sentient Computing

1. **Type and background**
Group/company: AT&T Research UK, Cambridge University Lab for Communications Engineering, UK
Time/manpower: Since 1998; development at AT&T Research UK closed in 2001.
Development focus: Research.
Research goals: To explore both the possibilities and the practical and social issues of ubiquitous computing and AR, based on the Active Bat ultrasonic, wireless building-wide tracking system.
Contact: Andy Hopper
Target environment: AR, Ubiquitous Computing.

2. **System description**
The Active Bat system uses small ultrasonic devices called *Bats* which are carried by users and attached to devices. It achieves a position accuracy of a few centimeters, but does not accurately measure orientation. The software architecture used within this project is called SPIRIT. The *Bat* system is controlled by a central server, which polls the Bats wirelessly, each in its own time slot. A scheduler can dynamically assign priorities to Bats that are moving frequently or which require a higher update rate, e.g., for a head-mounted display. The position of tracked objects is gathered on a central server. Object state, including position, is stored persistently in a relational database. Position information is directly forwarded to interested clients, without going through the database, so as to maintain interactivity. Clients then display information to the user, such as the location of colleagues. Mobile AR clients have been built, including a Laptop with HMD (with additional inertial tracking) and a PDA (which acts as a remote X terminal for an AR application running on the server).

However, most users do not use a mobile computer as a client, but use only the Bat itself; the Bat has a few buttons for input and can beep for output. Paper signs on the wall act as virtual buttons; by holding the Bat up to the sign and clicking the button on the Bat, the user activates a desired function.
Interesting aspects: The Active Bat system was (and probably still is) the largest tracking system of its accuracy.

3. **Underlying technology**
   Language: C++, Python (using CORBA).
   Network protocol: CORBA IIOP
   Supported platforms: Not specified.()
   Scalability: Actively used by 50 users with 200 bats on 3 building floors with 50 rooms.
   Underlying paradigm: Client/Server CORBA and database system

4. **Components**
   Types: The basic components are *CORBA objects* which correspond to real-world objects.
   Granularity: Large (people, walls, rooms, computers; each containing the full state, including position and orientation).
   Description: CORBA IDL.
   Instantiation: Not specified.
   Configuration: Not specified.
   Communication/lookup: Centralized (objects are located using a central server; communication is centralized as well).

5. **General information**
   Accessibility: Proprietary.
   Level of abstraction: The programmer works with CORBA objects.
   Modules and services: Position tracking, proximity detection, persistent storage, event notification.
   Suitable for: Sentient Computing is especially suitable for a fixed installation of a ubiquitous computing system within a medium-sized building.
   Key publications: [21, 40]

5.14 XWeb

1. **Type and background**
   Group/company: Brigham Young University, USA
   Time/manpower: Since 2000.
   Development focus: Research.
   Research goals: To address three main problems within the area of ubiquitous computing: first, the problem of device size vs. ease of interaction (e.g., PDA vs. desktop); second, the multimodal exploitation of interactive resources; and third, to obtain interactive control of a device by physical reference, e.g., pointing at it.
   Contact: Dan Olsen
   Target environment: Ubiquitous Computing.

2. **System description**
   At the heart of XWeb are *servers*, which are extended HTTP servers. They make an XML tree available to clients. Extensions to HTTP allow clients to access data in the XML tree and to subscribe to changes. *Services* are parts of a server's XML tree. Services can represent real-world entities, such as a temperature control. *Clients* are devices with which the user access services, e.g., a desktop, PDA or speech system. Using clients, the user can change values in a service, e.g., adjust the temperature. To allow several clients to work on the same task, clients can *subscribe* to a service. Thus, if a user changes a value within a service using one client, the other clients reflect this change as well. Each client is dynamically assigned

to a user *session*. A user's session consists of a reference to the service a user is interacting with, as well as a reference to the current *interactor*, which shows which part of a service the user has navigated to. XWeb supports two interaction metaphors for device management: *join* and *capture*. Users may *join* a service, or the active service of a specific client. For example, a user with a laptop and PDA may join the service that is being used in a meeting room, and both devices will display the same information as the other users' devices. A user may also *capture* a device, bringing it into his current session. For example, a user with a laptop may capture a projection display in the meeting room to show other users (who are not currently using the same service) some information.
Interesting aspects: The architecture of XWeb was designed with particularly simple user interaction in mind.

3. **Underlying technology**
Language: Not specified.
Network protocol: Extended *HTTP*.
Supported platforms: Not specified.
Scalability: Large scale; several rooms or building.
Underlying paradigm: Client/server, web technology.

4. **Components**
Types: *Services* on servers; *clients*.
Granularity: Medium (real-world controllable objects, e.g., thermostat; interaction devices).
Description: Services are accessible via XML.
Instantiation: Not specified.
Configuration: Not specified.
Communication/lookup: Lookup not specified; bidirectional Client/Server communication with multiple servers.

5. **General information**
Accessibility: Not specified.
Level of abstraction: The programmer works with XML trees.
Modules and services: Several displays and interaction devices, e.g., glove, pen, PDA, desktop, laptop.
Suitable for: XWeb is especially suitable for collaborative ubiquitous computing with a fixed infrastructure
Key publications: [42]

6 Conclusion

The sheer number of systems listed in this survey as well as its steady increase over time (Figure 2) shows the growing interest in research towards the vision of ubiquitous computing. Looking at each system more closely, many quite different approaches, each with its specific goals and strengths, have been followed in parallel, with equal justification. We have therefore tried to avoid personal judgments of the systems, presenting only facts available from scientific publications, web pages and personal communication with the authors.

Figure 3 shows on a world map, where the respective work was done. On this map some areas seem strikingly neglected, particularly Asia, especially Japan. The authors feel that there must be additional work in these regions which didn't surface in their literature survey. This might be due to the fact that it was published in different conferences or journals. In this sense, readers must keep in mind that the overview given in this article might be biased towards Europe and the U.S.

Nevertheless, the authors hope to provide a good starting point for researchers who need to use or develop infrastructures for research in ubiquitous computing. Furthermore, they hope to

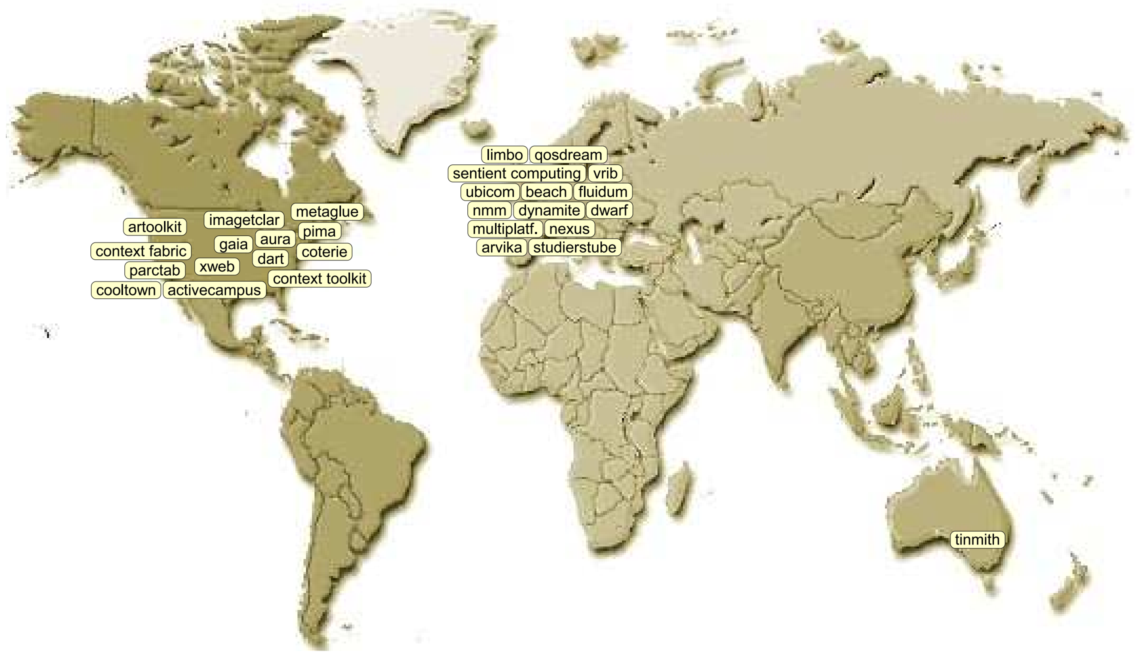

Figure 3: A geographic map of the systems discussed in this survey.

promote the exchange of ideas between research groups, and the different research communities involved in various aspects of making the vision of ubiquitous computing become reality.

7 Acknowledgments

We thank all researchers who have provided input for this survey beyond what was available in research papers. This work has been funded by the German Research Council (DFG) within the FLUIDUM project, the media informatics group at Munich University, Germany, the Chair for AI at Saarland University and the German Research Center for Artificial Intelligence, Saarbrücken, Germany, and the chair for applied software engineering at the Technical University of Munich, Germany. We thank our reviewers for valuable suggestions.

References

[1] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Deborra Zukowski. Challenges: an application model for pervasive computing. *Proceedings of 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, Aug. 2000.

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

[3] R. Bjornson, N. Carriero, D. Gelernter, T. Mattson, D. Kaminsky, and A. Sherman. Experience with linda. Technical Report YALEU/DCS/TR-866, Yale University, Department of Computer Science, Yale University, New Haven, Connecticut, U.S., 1991.

[4] Bernd Brügge, Asa MacWilliams, and Thomas Reicher. Software architectures for augmented reality systems – report to the ARVIKA consortium. Technical Report TUM-I0410, Technische Universität München, July 2004.

[5] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, A System of Patterns.* John Wiley & Sons Ltd, Chichester, England, 1996.

[6] Andreas Butz, Tobias Höllerer, Steven Feiner, Blair MacIntyre, and Clifford Beshers. Enveloping users and computers in a collaborative 3D augmented reality. In *Proceedings of the International Workshop on Augmented Reality IWAR '99*, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264, 1999. IEEE Computer Society Press.

[7] Deborah Caswell and Philippe Debaty. Creating web representations for places. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 114–126. Springer-Verlag, 2000.

[8] Renato Cerqueira, Carlos Cassino, and Roberto Ierusalimschy. Dynamic component gluing across different componentware systems. In *Proceedings of the International Symposium on Distributed Objects and Applications*, page 362. IEEE Computer Society, 1999.

[9] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The metaglue system. In Paddy Nixon, Gerard Lacey, and Simon Dobson, editors, *1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, pages 201–212, Dublin, Ireland, December 1999. Springer-Verlag.

[10] N. Davies, S. Wade, A. Friday, and G. Blair. Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications. In *Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97)*, pages 291–302, Toronto, Canada, May 1997.

[11] Anind K. Dey, Daniel Salber, and Gregory 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.

[12] Christoph Endres. Towards a Software Architecture for Device Management in Instrumented Environments. In *Adjunct Proceedings of UbiComp–The Fifth International Conference on Ubiquitous Computing*, pages 245–246, Seattle, Washington USA, October 2003. Doctoral Colloquium.

[13] Christoph Endres. *A Software Architecture for Device Management in Instrumented Spaces.* PhD thesis, Saarland University, Germany, 2005. To appear.

[14] Steven Feiner, Blair MacIntyre, Tobias Höllerer, and Tony Webster. A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. In *Proc. ISWC '97 (First IEEE Int. Symp. on Wearable Computers), October 13-14, 1997, Cambridge, MA.* IEEE press, 1997.

[15] A. Friday, G. Blair, K. Cheverst, and N. Davies. Extensions to ANSAware for advanced mobile applications. In *Proceedings of the 1st International Conference on Distributed Platforms ICDP, Dresden, Germany, 27 February-1 March 1996.*

[16] Wolfgang Friedrich, editor. *ARVIKA – Augmented Reality für Entwicklung, Produktion und Service.* Publicis, Erlangen, 2004.

[17] Maribeth Gandy, Steven Dow, and Blair MacIntyre. Prototyping Applications with Tangible User Interfaces in DART, The Designers Augmented Reality Toolkit. In *Positional Paper at Toolkit Support for Interaction in the Physical World Workshop at the IEEE Pervasive Computing*, April 2004.

[18] Al Geist, Adam Beguelin, Jack Dongorra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderman. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press, Cambridge, MA, 1994.

[19] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[20] William G. Griswold, Robert Boyer, Steven W. Brown, and Tan Minh Truong. A component architecture for an extensible, highly integrated context-aware computing infrastructure. In *Proceecings of the International Conference on Software Engineering*, pages 363–372, Portland, Oregon, 2003. IEEE Computer Society.

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

[22] Thomas Heider and Thomas Kirste. Supporting goal-based interaction with dynamic intelligent environments. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002*, pages 596–600, Lyon, France, July 2002. IOS Press.

[23] Michael Hellenschmidt and Thomas Kirste. SodaPop: A Software are Infrastructure Supporting Self-Organization in Intelligent Environments. In *Proceedings of the 2nd IEEE International Conference on Industrial Informatics, INDIN'04*, Berlin, Germany, 2004.

[24] Gerd Herzog, Heinz Kirchmann, Stefan Merten, Alassane Ndiaye, Peter Poller, and Tilman Becker. MULTIPLATFORM Testbed: An Integration Platform for Multimodal Dialog Systems. In Hamish Cunningham and Jon Patrick, editors, *HLT-NAACL 2003 Workshop: Software Engineering and Architecture of Language Technology Systems (SEALTS)*, pages 75–82, Edmonton, Canada, 2003. Association for Computational Linguistics.

[25] Gerd Herzog, Alassane Ndiaye, Stefan Merten, Heinz Kirchmann, Tilman Becker, and Peter Poller. Large-scale Software Integration for Spoken Language and Multimodal Dialog Systems. *Natural Language Engineering*, 10, 2004. Special issue on Software Architecture for Language Engineering, to appear.

[26] J. I. Hong and J. A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the Second International Conference on Mobile Systems, Applications and Services*, Boston, MA, 2004.

[27] Roberto Ierusalimschy and Luiz Henrique de Figuereido andWaldemar Celes. Lua: An Extensible extension language. In *Software: Practice and Experience*, 1996.

[28] Hirokazu Kato and Mark Billinghurst. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99), San Francisco, USA*, 1999.

[29] Tim Kindberg and John Barton. A Web-Based Nomadic Computing System. Technical Report HPL-2000-110, Internet and Mobile Systems Laboratory, HP Laboratories, Palo Alto, August 2000.

[30] Andreas Klüter, Alassane Ndiaye, and Heinz Kirchmann. Verbmobil From a Software Engineering Point of View: System Design and Software Integration. In Wahlster [57], pages 635–658.

[31] R. L. Lagendijk. Ubiquitous communications (ubicom) - updated technical annex 2000. Technical report, Ubiquitous Communications Program TU-Delft, Jan. 2000.

[32] 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.

[33] Blair MacIntyre and Steven Feiner. Language-level support for exploratory programming of distributed virtual environments. In *Proc. UIST '96 (ACM Symp. on User Interface Software and Technology), Seattle, WA, November 6-8*, pages 83 – 95, 1996.

[34] Blair MacIntyre and Steven Feiner. A distributed 3D graphics library. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 361–370. ACM Press, 1998.

[35] Asa MacWilliams, Thomas Reicher, and Bernd Brügge. Decentralized coordination of distributed interdependent services. In *IEEE Distributed Systems Online – Middleware '03 Work in Progress Papers*, Rio de Janeiro, Brazil, June 2003.

[36] 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)*, October 2003.

[37] Marco Lohse and Michael Repplinger and Philipp Slusallek. Session Sharing as Middleware Service for Distributed Multimedia Applications. In *Interactive Multimedia on Next Generation Networks, Proceedings of First International Workshop on Multimedia Interactive Protocols and Systems, MIPS 2003*, volume 2899 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2003.

[38] Marco Lohse and Philipp Slusallek. Middleware Support for Seamless Multimedia Home Entertainment for Mobile Users and Heterogeneous Environments. In *Proceedings of The 7th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA)*, pages 217–222. ACTA Press, 2003.

[39] Hani Naguib, George Coulouris, and Scott Mitchell. Middleware support for context-aware multimedia applications. In *DAIS*, 2001.

[40] Joseph Newman, David Ingram, and Andy Hopper. Augmented reality in a wide area sentient environment. In *Proceedings of the International Symposium on Augmented Reality (ISAR)*, October 2001.

[41] Daniela Nicklas, Matthias Großmann, Thomas Schwarz, Steffen Volz, and Bernhard Mitschang. A model-based, open architecture for mobile, spatially aware applications. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 117–135. Springer-Verlag, 2001.

[42] Dan R. Olsen, Jr., S. Travis Nielsen, and David Parslow. Join and capture: a model for nomadic interaction. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 131–140. ACM Press, 2001.

[43] Charles Owen, Arthur Tang, and Fan Xiao. ImageTclAR: A blended script and compiled code development system for augmented reality. In *Proceedings of the International Workshop on Software Technology for Augmented Reality Systems (STARS)*, October 2003.

[44] W. Pasman and F. W. Jansen. Distributed low-latency rendering for mobile ar. In *Proceedings of the International Symposium on Augmented Reality (ISAR)*, October 2001.

[45] Brenton Phillips. Metaglue: A programming language for multi-agent systems. M.Eng thesis, MIT, 1999.

[46] W. Piekarski and B. H. Thomas. An object-oriented software architecture for 3D mixed reality applications. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, October 2003.

[47] Gerhard Reithmayr and Dieter Schmalstieg. OpenTracker – an open software architecture for reconfigurable tracking based on XML. Technical report, TU Wien, 2000.

[48] Manuel Roman, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, Oct-Dec 2002.

[49] D. Schmalstieg, A. Fuhrmann, G. Hesina, Zs. Szalavari, L. Miguel Encarnação, M. Gervautz, and W. Purgathofer. The Studierstube Augmented Reality Project. *Presence*, 11(No.1), 2002.

[50] R. Schönfelder, G. Wolf, M. Reeßing, R. Krüger, and B. Brüderlin. A pragmatic approach to a VR/AR component integration framework for rapid system setup. In *Proceedings of the Paderborn Workshop "Augmented und Virtual Reality in der Produktentstehung"*, Paderborn, 2002.

[51] João Pedro Sousa and David Garlan. AURA: An Architectural Framework for User Mobility in Ubiquitous Computing Environments. In Bosch, Gentleman, Hofmeister, and Kuusela, editors, *Software Architecture: Design, Development, and Maintainance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture)*, pages 19–43. Kluwer Academic Publishers, August 2002.

[52] João Pedro Sousa and David Garlan. The Aura Software Architecture: an Infrastructure for Ubiquitous Computing. Technical Report CMU-CS-03-183, School of Computer Science, Carnegie Mellon University, Pittsburg, PA 15213-3890, August 2003.

[53] Peter Tandler. Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices. In *Proceedings of UbiComp 2001: Ubiquitous Computing*, number 2201 in LNCS, pages 96–115. Springer Verlag, Heidelberg, 2001.

[54] Peter Tandler. The BEACH application model and software framework for synchronous collaboration in ubiquitous computing environments. *Journal of Systems and Software*, 69(3):267–296, 2004.

[55] Peter Tandler. *Synchronous Collaboration in Ubiquitous Computing Environments*. PhD thesis, Technische Universität Darmstadt, Fachbereich Informatik, to be published in 2004.

[56] Russel M. Taylor, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. VRPN: a device-independent, network-transparent VR peripheral system. In *Proceedings of the ACM symposium on Virtual Reality Software and Technology 2001 (VRST-2001)*, pages 55–61, Banff, Alberta, Canada, 11 2001.

[57] Wolfgang Wahlster, editor. *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer, Berlin, 2000.

[58] Wolfgang Wahlster, editor. *SmartKom: Foundations of Multimodal Dialogue Systems*. Springer, Berlin, 2004. to appear.

[59] Roy Want, Bill Schilit, Norman Adams, Rich Gold, Karin Petersen, John Ellis, David Goldberg, and Mark Weiser. The PARCTAB ubiquitous computing experiment. Technical Report CSL-95-1, Xerox Palo Alto Research Center, March 1995.

[60] Mark Weiser. The computer for the 21st century. *Scientific American*, 3(265):94–104, 1991.