Ubiquitous Animated Agents for Augmented Reality

István Barakonyi Dieter Schmalstieg
Graz University of Technology, Institute of Computer Graphics and Vision
Inffeldgasse 16, Graz, A-8010 Austria
Email: { bara | dieter } @ icg.tu-graz.ac.at

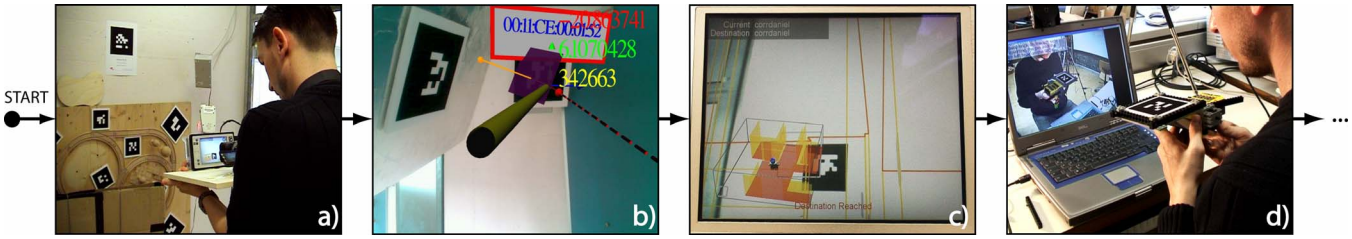

Figure 1: Going through a technician's to-do list with the Ubiquitous Technician application. This application seamlessly combines an indoor AR navigation system (a,c), an ultra-wideband calibration aid (b), and a machine maintenance application (d).

ABSTRACT

Most of today's Augmented Reality (AR) systems operate as passive information browsers relying on a finite and deterministic world model and a predefined hardware and software infrastructure. We propose an AR framework that dynamically and proactively exploits hitherto unknown applications and hardware devices, and adapts the appearance of the user interface to persistently stored and accumulated user preferences. Our framework explores proactive computing, multi-user interface adaptation, and user interface migration. We employ mobile and autonomous agents embodied by real and virtual objects as an interface and interaction metaphor, where agent bodies are able to opportunistically migrate between multiple AR applications and computing platforms to best match the needs of the current application context. We present two pilot applications to illustrate design concepts.

CR Categories: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems - Artificial, Augmented, and Virtual Realities; I.3.6 [Computer Graphics]: Methodology and Techniques - Interaction techniques.

Keywords: augmented reality, ubiquitous computing, animated agents, proactive computing, context adaptation

1 INTRODUCTION

Most of today's Augmented Reality (AR) systems operate as passive information browsers relying on a predefined, static set of hardware and software components. The only dynamic element of such systems is usually the world model, updated manually to store information about the physical environment and thus offer an interface between the real and virtual world. These world models are finite and deterministic, requiring application developers to exhaustively enumerate all possibilities.

In contrast, research on Ubiquitous Computing (Ubicomp) revolves around building frameworks for the seamless and automatic integration of diverse hardware and software components, making these disparate resources easily accessible as standard services. One research agenda is therefore that AR systems need to be complemented with Ubicomp techniques to effectively exploit distributed resources such as software services, sensors, and output devices.

Somewhat related to Ubicomp, Autonomous Agent research is concerned with delivering "smart" distributed software components and services for end users, particularly in the domain of interface agents. Autonomous agents are proactive software entities facilitating informed decision-making. These decision-making capabilities are not necessarily based on a deep understanding of the problem semantics, yet allow the agent to deliver a useful function in a complex, heterogeneous environment. Animated agents are a specialization of autonomous agents possessing visual, often anthropomorphic representations in a simulated virtual world.

In previous work [1], we have shown how to construct animated agents in AR (AR agents), which have a physical as well as a virtual part in their input and output modalities. AR agents can thus autonomously bridge the gap between the real and virtual part of mixed reality. However, the AR agent design presented in our earlier work suffered from some limitations:

(1) The agent's brain was hard-coded and not programmable on the fly. As a consequence, the agent operated with a static world model. While sensor information and user input provided live updates to the attributes of the world model, the structure of the world was defined in advance. The agent was thus programmed to work with a specific application and setting, and unable to adapt to new scenarios or react to unforeseen situations. In other words, the AR agent lacked typical capabilities of Ubicomp.

(2) The agent's behavior was mostly driven by the runtime system's flow of events and had only limited autonomy. The agents could respond to user input and other activities observed in the system but could not derive any higher-order strategies that truly qualify as autonomous behavior. In combination with the static world model mentioned above, the set of different behaviors that the agent could exhibit was limited and deterministic.

In this paper we present an improved design for a *ubiquitous AR agent* (or UbiAgent for short), which overcomes most of these limitations. The UbiAgent works similarly to the AR agent, however, its behavior is not only event-driven but also proactive. Its decisions are based on a self-contained reasoning engine relying on a knowledge base that is externalized in a persistent XML database. Thus it becomes easy to influence the agent at runtime by updating its knowledge base.

The knowledge base is designed as a shared memory area, so that multiple UbiAgents and other distributed software

components representing applications and real world objects can exchange messages. New software components can be dynamically added to this knowledge base, and the UbiAgent can learn to communicate with them through a standardized interface.

The persistency of the knowledge base allows the UbiAgent to preserve its state and preferences over time, so that it can opportunistically migrate from one networked AR environment to another, following a user around. Since the UbiAgent can also change its appearance in response to the current environmental conditions and these changes can be stored and retrieved on a per-user basis, it is capable of a behavior that we call *multi-user interface adaptation*.

It is important to note that we do not claim that the UbiAgent is intelligent in the classic artificial intelligence (AI) sense; our work is more influenced by what is generally described as ambient intelligence [2]. The main contribution of our work is a set of new techniques to create adaptive user interfaces for AR applications.

Our paper first discusses related work on adaptive AR. Then principles of our framework design are introduced including implementation details. Finally two applications are presented to illustrate design concepts (see Figure 1 for images illustrating the application described in Section 5.2) and serve as pilot studies.

2 Related Work

In this section we provide an overview of techniques how augmented reality systems in related research projects facilitate adaptive behavior by tailoring their user interface to dynamically changing context information.

2.1. Information Filtering

The simplest form of automatic adaptation of AR content to current application context is information filtering based on a spatial or semantic world model. The primary objective of information filtering is to avoid cluttering displays by an unnecessarily large number of visual elements, and thus overwhelming and confusing users with unimportant information.

Classic computer graphics applications [3] apply spatial filtering based on simple context elements such as distance and visibility. These filters reduce computational and cognitive workload by culling away irrelevant visual objects or reducing their level of detail. A typical AR example based on the latter approaches is the work of Bane and Höllerer [4], who developed interactive tools to enhance building visualization in a mobile AR setup.

While spatial filters are efficient tools for enhancing spatial tasks such as indoor/outdoor navigation, AR applications need to frequently consider a larger and more diverse set of context elements. The KARMA system [5] employs a rule-based illustration generation system to exploit user viewpoint, object pose and communicative goals for efficient information visualization in an AR-based machine maintenance scenario. Although KARMA proved to be suitable for replacing manuals for a small-scale repair task, rule-based generation of visual augmentations for larger and more complex systems suffers from scalability problems. Julier et al. developed a hybrid approach [6] for their mobile AR system: a spatial model is used to prefilter visual elements to reduce input information for a rule-based filter component.

2.2. Adaptive User Interface Components

Besides the rendering engine filtering out data, other system components may also actively adapt their behavior to dynamically changing context information. A typical example is the UbiTrack project [7], which eliminates dependencies on specific sensors by dynamically incorporating data arriving from a heterogeneous network of distributed sensors. Kaiser et al. [8] developed an immersive AR environment that retrieves context information on demand to disambiguate multimodal interaction by estimating user intention from deictic speech utterances and 3D gestures.

The hybrid user interface developed by Benko et al. [9] uses 2D and 3D gestures to switch between interaction contexts to determine the target display and privacy factors in a multi-display environment. The properties of the current target display impose dimensional constraints onto the available interaction methods in the user interface; for instance, a touchscreen panel permits only 2D gestures, while the immersive work environment of a head-mounted display demands 3D gestures.

2.3. User Interface Migration

A cross-dimensional interface is a notable example for a special type of adaptive component: the user interface migration controller. This component is responsible for interface migration between computing platforms with different characteristics. Full or partial migration of user interface elements between devices and displays allows the selection of the most suitable environment for presenting application information [10]. For instance, a PDA or mobile phone offers only limited rendering and interaction capabilities but enables users to roam a large area without interruptions in their work flow. On the other hand, monitors and projection screens are stationary but support a shared view and richer presentation tools. Smooth transitions across multiple devices running the same distributed application mitigate the seam between platform boundaries and thus increase productivity.

Further migration examples in AR include the playful SHEEP application scenario [11] that employs 3D gestures and tangible interaction to initiate the transfer of a virtual sheep model between multiple stationary and mobile displays. Schmalstieg et al. [12] created a shared collaborative workspace based on a distributed shared scene graph that enables the migration of applications between hosts. Their work addresses ad-hoc collaboration and load balancing for AR environments.

The Augmented Surfaces project [13] applies user interface migration techniques in a spatially continuous augmented physical workspace spanning multiple portable computers and fixed displays. Devices are identified by 2D fiducial markers, the relative pose of which triggers the migration of application objects. A new device can be added to the workspace if it implements an object serialization interface and is tagged by a unique marker.

The EMMIE framework [14] introduces a hybrid user interface for AR systems enabling information management using a wide range of hardware devices. EMMIE's environment manager component addresses the needs of Ubicomp by providing techniques such as mixed reality interaction and privacy management to organize virtual information on several displays shared by multiple users.

Augmented Surfaces and EMMIE implement ideas in a way that is conceptually closest to our work, however, neither EMMIE nor the Augmented Surfaces framework includes concepts such as proactive interface adaptation, persistent preference storage, and resource discovery.

2.4. Software Agents in AR

All information filtering and adaptive component approaches share a significant shortcoming: the set of context and world model elements must be finite and deterministic to trigger

appropriate system reaction at any time. If a novel system component or unknown external application appears and produces an unsupported type of information, then new filters and components must be created to take advantage of the hitherto unknown data types. For instance, an interface knowing only about spatial relationships of application objects has to be taught how to handle information such as user profile, application history or hardware capabilities.

Although interface agents and autonomous software components generate much controversy in the HCI community [15], we argue that AR systems can benefit from software agent technology. Agents are designed to be independent from applications they are embedded into, enabling their employment in diverse application environments without reprogramming the applications' core functionalities. Moreover, the incorporation of flexible high-level context elements such as application goal and user interest may more efficiently cope with the indeterministic nature of augmented physical environments than explicit direct manipulation techniques.

Software agents have been present in AR applications so far in the form of animated anthropomorphic characters. The early ALIVE system [16] exhibits a virtual animated character composited into the user's real environment that responds to human body gestures on a large projection screen. The Welbo project [17] features an immersive setup, where an animated virtual robot assists an interior designer wearing an HMD. MacIntyre et al. [18] place prerecorded video-based actors into an AR environment to create an interactive theater experience. Cheok et al. [19] also experiment with mixed reality entertainment with live captured 3D characters, which enable telepresence of real people within a virtual or augmented reality setting. Cavazza et al. [20] place a live video avatar of a real person into a mixed reality setting, and interact with a digital storytelling system with body gestures and language commands. Balcisoy et al. [21] employed virtual humans in mixed reality as collaborative game partners, while Vacchetti et al. [22] used a virtual lifelike character in a training scenario for real factory machinery.

Except the ALIVE system, all aforementioned research projects feature characters that are interface agents with little or no autonomous behavior, relying on explicit user input for their actions and thus representing only advanced forms of traditional command line interfaces enhanced by rich multimedia elements. There is a clear distinction between interface agents [23] operating as assistants for a direct manipulation interface and autonomous agents [24] acting parallel with the user to carry out delegated tasks being uninteresting or time consuming. We combine the advantages of both approaches into an autonomous interface agent [25] that executes tasks and provides feedback without constant attention and explicit commands while monitoring the user's environment and actions.

2.5. Mobile Agents

Unlike desktop agents that are limited to operate in the 2D world of a computer screen, agents in AR may move in the user's physical environment using all 6 degrees of freedom. With the simultaneous use of various stationary and mobile devices AR environments offer not only the freedom of a single three-dimensional physical space mapped to a display but several interconnected spaces. Thus autonomous interface agents increase their mobility and gain another output modality, as the current pose and choice of display may both carry an important message for users.

Kotz and Gray [26] use the term *mobile agent* for autonomous software components that have the ability to "transfer and reproduce" themselves on various networked computing devices. By equipping AR agents with mobile characteristics, they are no longer bound to a single, statically configured application and output device but may opportunistically migrate to and take advantage of other platforms more favorable for the agent's current needs.

Recent advances in hardware and software technology for portable devices such as PDAs and smartphones have eliminated previously serious constraints on the visual representation of migratable agents. While the early C-MAP system [27] visualizes its context-aware virtual museum guide as a sequence of static images, Wagner et al. [28] presents a similar scenario in AR with a full-fledged virtual 3D character exhibiting reactive behavior on a consumer PDA. The PEACH system [29] has experimented with visualization techniques using an animated cartoon character to preserve the continuity of an animated presentation spanning multiple displays. The embodied mobile agents of the Virtual Raft project [30] appear to "jump" between tablet PCs carried by participants of a playful museum exhibition.

Besides our work, the Agent Chameleons framework [31] has been the only project to date allowing agents to seamlessly travel between real and virtual bodies while being controlled by a central control logic. Similarly to UbiAgents, Agent Chameleons are also based on a BDI agent architecture (see Section 3.5 for details), however, their system design lacks essential properties of ubiquitous AR systems such as application encapsulation, persistency, and dynamic agent platform management.

3 Design Principles

After the overview of related agent technologies, we can position our UbiAgent approach within the research domain of software agents. As Figure 2 illustrates, UbiAgents combine advantageous characteristics of three separate agent research areas to create a framework satisfying the needs of adaptive AR interfaces. This section introduces principles our framework design is based on.

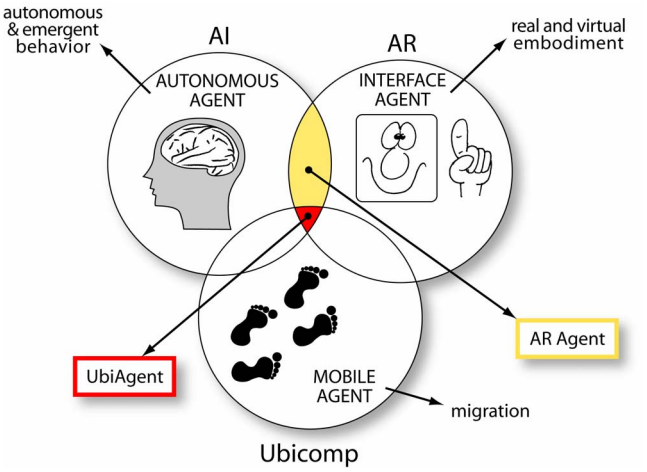

Figure 2: UbiAgents within software agent research

3.1. Building upon AR agents

Our previous work [1], the AR Puppet project integrated interface agents with autonomous agents to form AR agents, which are "smart" software components embodied by real and virtual objects operating in the user interface of AR systems. The

capabilities of AR agents were demonstrated by a machine maintenance application, where a virtual animated repairman assisted an untrained user to assemble and maintain a real LEGO® Mindstorms robot. The behavior engine of AR agents relied on a finite state machine, where states represented sets of behavior elements (animations, sounds, etc.) and transitions were triggered by events coming from sensors deployed in the real and virtual world.

The virtual repairman agent operated independently from the robot maintenance application. It relied only on its own observations of application attributes and user input to generate an animated presentation of robot features without explicit user guidance. However, the state machine and the world model could not be reconfigured at run-time. States and transitions were hard-coded, and the structure of the world model was manually configured before application start-up.

3.2. Increasing Mobility

To highlight key differences between UbiAgents and AR agents, we revisit the LEGO maintenance example in the following sections. UbiAgents enhance AR agents with mobile agent technology and migration capabilities by extending the previous finite state machine approach. Each state in the UbiAgent's state machine is now associated with certain requirements on the current software and hardware environment.

In the LEGO maintenance scenario the current robot construction step may require a minimum screen size and display resolution if it involves subtle visualization details of a complex engine. Other agent state preconditions may set a minimum CPU speed and maximum memory load for resource hungry animations. Portable computing devices may frequently demand agents to check the battery level and the wireless network status to avoid abruptly losing important application data.

If the requirements are met, the agent proceeds with the execution of the behavioral action sequence assigned to the current state in the state machine. However, if the capabilities of the current agent platform prove inadequate, the agent searches for another environment providing more favorable conditions to complete its job. If such an environment is found, the agent opportunistically migrates to it and executes actions in its new "home". Returning to the LEGO example, the virtual repairman may move to a bigger screen located near the user and continue to explain the robot's internal structure there, if the agent finds the current display too small for the current construction step.

Migration is signalized by a special behavioral sign such as an animation sequence, text warning or sound alert to make the user aware of the migration action or to instruct the user to prevent migration within a certain grace period. In case of the preventive scenario, the agent proactively suspends its current activities and advises the user to charge her PDA or set the screen resolution of the PC monitor to match minimum requirements. After the grace period expired without appropriate changes in the local workspace, the agent migrates to its new preferred environment and resumes its actions. Although migration causes an interruption in the application flow, this temporary break in the continuity of user interaction is invoked in favor of a more efficient work environment, shortening overall interaction times and increasing the quality of information visualization.

Migratable user interfaces demand that dominant interface properties are preserved during and after migration to bridge the spatial and cognitive gap between disjoint workspaces. The user has to create a mental link between the old and new workspaces, thinking that the same virtual assistant continues to aid her work with the augmented LEGO robot, even if it migrated to a projection screen from a local display to increase its public exposure.

The agent should appear to continue its task exactly at the point where it left off before migration. Beside temporally continuous agent behavior, visual agent appearance also frequently needs to remain unchanged across multiple agent environments by migrating respective 3D models, textures, color schemes, spatial arrangement etc. together with the mental state. Nevertheless, this is not a general requirement since in some cases the agent may deliberately choose a different visual representation. For example, a simple arrow may replace the pointing gesture of the full-body animated repairman of the robot maintenance application to avoid occluding subtle details of tiny mechanical robot parts. Similarly, the agent may choose to occupy a physical body taking advantage of sensors and actuators affecting the real world. For instance, the robot maintenance scenario can manipulate the real robot instead of a virtual robot model simulation, if the physical counterpart is available.

Agent migration and the preservation of agent attributes and mental state during migration necessitate the use of a central control logic that supervises agent embodiments. We call this component the *agent brain* being in charge of controlling multiple agent representations or *agent bodies*. The agent brain relies on a persistent information storage, where agent and workspace attributes can be saved and recalled.

3.3. Expect the Unexpected

Let us imagine that we buy a new microwave oven for our kitchen and want to employ an AR system to explain its operation. With current classic AR software design we would use a standalone application tailored to the explanation of our specific microwave oven model. We cannot get the already familiar animated repairman to introduce us our new household item instead of a LEGO robot, as this AR agent has never "seen" a microwave oven before and thus it does not know how to present it.

The UbiAgent framework teaches the old dog new tricks: we equip AR agents with capabilities to adapt to and work with hitherto unknown applications. Consequently, if we enhance the aforementioned household scenario with UbiAgent components, the microwave oven application becomes part of the dynamic world model of the new UbiAgent-based repairman character. If the application generates a request calling for an animated presentation of its typical features, the repairman agent migrates to the new environment and starts the explanation.

UbiAgents encapsulate AR applications as black boxes hiding implementation details and communicating via relevant input and output attributes with the outside world. The black box interface maps private, internal application state to public attributes based on a well-defined schema. Any agent "understanding" this schema is able to automatically establish communication with the application, deduce application state information by monitoring these attributes, and influence application behavior by modifying attribute values. Figure 3 provides illustration for the concept about schemas and application encapsulation.

The diversity of AR application domains demands the creation of multiple schemas. Systems acting in the fashion of digital manuals need a schema enabling the presentation of a range of household devices, computing equipment and furniture, while indoor and outdoor navigation systems necessitate a schema for the encapsulation of parts of the physical environment such as floors, offices, streets, and buildings.

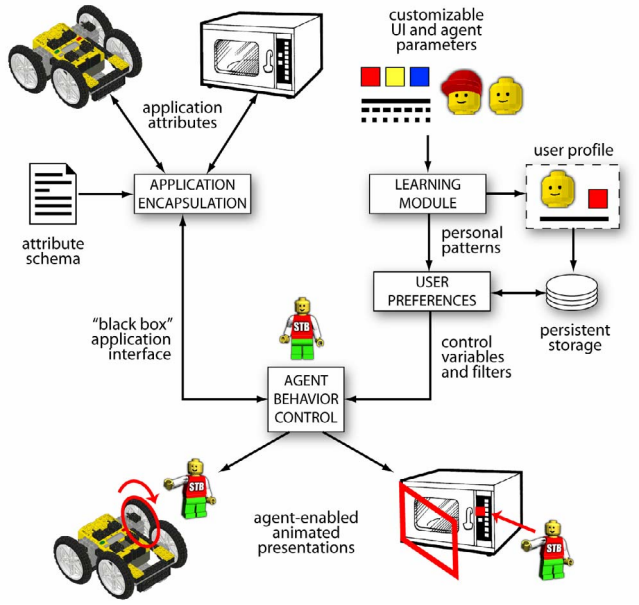

Figure 3: Application encapsulation with schema and adaptive user interface personalization

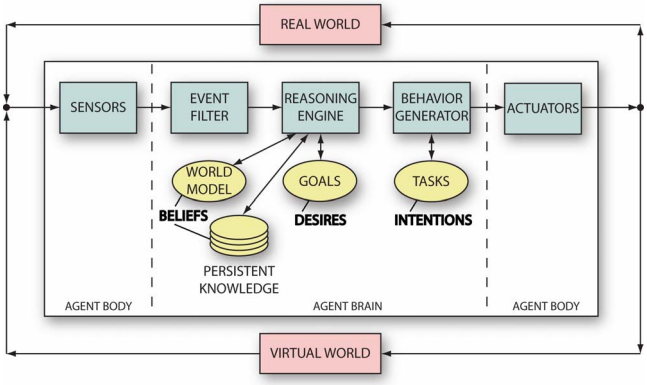

Figure 4: UbiAgent structure based on the BDI model

Multi-purpose agents need to understand several schemas to let the same agent work as a virtual tour guide or animated technician on demand. When a hitherto unknown application appears in the system and a UbiAgent wants to communicate with it, the agent first checks its schema. If the agent "speaks the language" of the schema, it includes the application in its control loop and reacts accordingly to attribute changes.

3.4. Multi-user Interface Adaptation

Users favor customizable interfaces over fixed ones. People have diverse preferences for the color, size, spatial arrangement, and numerous other style elements of user interface components, including accessibility features for the disabled.

Present AR systems offer offline tweaking of variables in parameterized user interfaces to dynamically change interface appearance, however, customization information is only considered in the current session without being stored in a persistent memory, ignoring adaptive and multi-user concepts. As illustrated by Figure 3, the UbiAgent framework includes a persistent database to store user preferences observed and accumulated by a learning module for future application sessions. The learning module captures typical patterns in the way users set interface customization parameters and stores them in a personal user interface profile in the database. This personalization profile follows users around while they are working with multiple distributed applications running on various computing devices.

A nearsighted user working with the previous section's machine presentation scenario would always enlarge objects to notice small details in the LEGO robot's mechanical structure. The learning module perceives this personal customization pattern and stores it in the database. The next time this particular user runs the LEGO maintenance application, all objects would be automatically enlarged based on the previously observed and stored preferences. When switching over to the microwave oven application, the user would find the virtual objects' default size already scaled up, saving hitherto efforts to tailor application interfaces to the user's taste and convenience.

The UbiAgent framework is based on a fast and robust database that enables storing and recalling preferences on demand for a large number of users, thus enhancing AR systems with *multi-user interface adaptation* capabilities. Identification of individual users relies on a unique user ID associated with personal devices such as PDAs or tablet PCs, or based on user accounts for shared public computers.

3.5. Beliefs, Desires, Intentions

Our framework follows the belief-desire-intention (BDI) model [32] for the implementation of the agent's reasoning mechanism. This model is not only one of the most well-known approaches for practical reasoning agents with a substantial research corpus, but is also highly suitable for dynamic and uncertain environments such as AR systems. Figure 4 depicts the BDI model-based structure of UbiAgents.

Beliefs represent the agent's current knowledge of the real world (such as the estimated pose and internal attributes of application objects) mapped to an internal world model. Since the world model represents only a potentially imperfect local view of the physical and virtual world, it needs to be regularly updated by measurements coming from sensors in the real and virtual environment. The database caches the current world model state between measurements and stores persistent information such as user preferences, application attributes, and agent properties.

Desires stand for agent goals associated with a desired end system state. They represent high-level concepts in the UbiAgent's brain subordinating user interface components to adapt their behavior to achieve goals as quickly as possible. UbiAgents work towards their goals by carrying out tasks or *Intentions* using actuators in the real and virtual world. The currently executed tasks are constantly reevaluated to verify whether they are efficiently advancing the system towards the end state. The system may reconsider its decisions in case of inadequate progress, and kill suboptimal tasks while starting new, more promising ones.

According to Georgeff et al. [33] adaptive, goal-oriented systems offer a superior performance compared to task-oriented systems in dynamic environments requiring automatic recovery from erroneous situations. In task-oriented systems each task strives to achieve a local optimum without remembering the purpose of its execution. In ubiquitous AR environments, where failures and suboptimal working conditions are inevitable due to

the simultaneous use of multiple interconnected hardware and software components, a flexible and adaptive software architecture is needed to effectively tackle issues such as computer crashes, load balancing, and resource discovery.

In UbiAgents, goals represent a combination of desired application and agent states, for instance "the repairman presents the operation of the LEGO robot's light sensor". This high-level goal is decomposed into subgoals or plans equivalent to application attribute changes and animated agent action sequences such as "proceeding the maintenance application to the step where the light sensor is activated on the real robot, and superimposing sensor measurement values over the real sensor while the agent explains the sensor's operation".

Plans are converted into concrete agent tasks that are executed by actuators available in the current agent environment. In AR actuators can be physical as well as virtual. Typical examples for virtual actuators include animation engines controlling 3D models and virtual characters, 2D text messages, sound players, text-to-speech engines, etc. Common physical actuators involve stationary and mobile computers with limited resources such as CPU speed and memory size, fixed and portable displays with a predefined size and resolution, audio speakers, and electric motors and control systems of mechanical engines.

Before task execution, the UbiAgent framework checks whether actuators required for the next task are available at the current agent platform. For instance, the light sensor scenario requires that the agent has access to an infrared communication port to exchange messages with the physical LEGO robot to let the repairman agent turn on the real light sensor during explanation. This step also requires a PC with a fast CPU and a large display for 3D visualization purposes, and a 6DOF tracking system to facilitate the correct overlay of virtual information on top of the real world.

If the desired sensors and actuators are missing or fail to meet the minimum requirements in the current agent environment, the UbiAgent consults its beliefs in the database storing persistent knowledge about all available agent platforms, and looks for a more suitable environment where it can migrate to and complete its current job.

4 UBIAGENT COMPONENTS

In this section we present the UbiAgent framework components and explain their functions with references to the design principles described in the previous section. Figure 5a shows a diagram with all UbiAgent entities and their relationships.

Each UbiAgent consists of an *agent brain* and one or more *agent bodies*. The agent brain serves as a control logic and reasoning engine controlling global agent behavior. The agent body is a local representation of the agent brain and an embodiment of the agent. As UbiAgents operate in AR environments, they are allowed to possess real as well as virtual bodies, and thus appear to be integral parts of the user's physical surroundings. The agent body contains sensors observing the agent's real and virtual environment, and actuators affecting the physical and virtual world.

UbiAgents do not exist on their own. Similarly to symbiotic life-forms, they need a host to exploit for living while offering services for the host's benefit. In our framework we call this host environment a *habitat*, which is characterized by its *hospitability* attribute. The habitat's hospitability symbolizes the amount of "nutrients" available for digital symbionts such as UbiAgents, and combines diverse hardware parameters into a single value to describe the computational power of potential host machines.

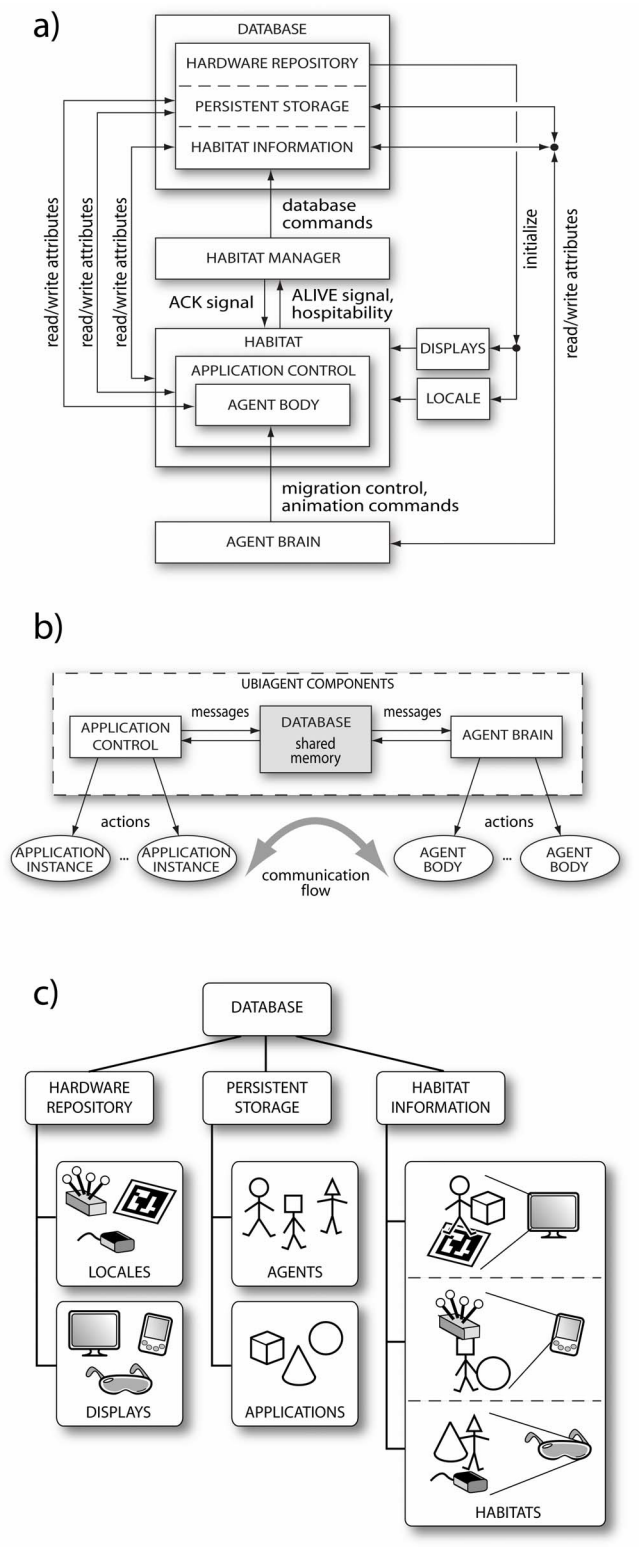

Figure 5: a) Entities and their relationships in the UbiAgent framework, b) Shared memory between agents and applications, c) Structure of the UbiAgent database

The single hospitability value hides irrelevant internal technical details of diverse computing platforms such as CPU load, available memory, or remaining battery power, yet allows agents to identify erroneous situations such as a crashing or overloaded device by their low hospitability value. Poor habitat conditions endangering agent operation trigger survival agent behavior that usually forces migration to another habitat. Graceful degradation of agent services is also possible as the availability of resources declines.

Computing devices serving as agent habitats in AR must provide one or more *displays* and a network of tracking systems called a *locale* to support the superimposition of virtual information over the real world. Some parameters such as display size or the degrees of freedom of tracking data rarely change, therefore they are stored in a *hardware repository*. Repository information is manually updated by a technician when a new display device or tracking system is installed.

Habitats constantly update their current attributes in a *habitat information* storage, including pointers to applications they are hosting, references to agents currently embedded in the applications, and information about currently associated locales and displays to provide a quick overview of the present habitats' hardware and software infrastructure. Invariant display and locale parameters are loaded from the repository, while dynamic parameters such as screen resolution are updated by the respective embedded display or locale component.

As occasional interruptions in network connectivity and power failures of computing devices are inevitable, habitats need a specific component that detects their temporary inability to communicate with UbiAgents despite their presence in the agents' world model. This component is called the *habitat manager*, implementing autonomous functionalities such as low-level resource management of agent platforms, garbage collection, integrity checks in the database, and repairing broken connections between UbiAgent components. To prevent UbiAgents from attempting to communicate with a non-responsive agent platform, the *habitat manager* removes habitats from the world model unless they periodically send an "alive" signal. The habitats keep pinging the habitat manager until it replies with an "acknowledge" message, which reassures faultless communication with UbiAgents until the next periodic "alive" signal is due.

4.1. Shared Agent and Application Memory

The dynamic nature of AR environments makes constant and reliable communication between framework objects crucial. Thus the *database* component plays an eminent role for UbiAgents. The database not only stores the hardware repository, current habitat information, and persistent UbiAgent component attributes, but also serves as a shared memory between agents and applications (see Figure 5b). Components can register observers in the database, which are requests for notifications about changes in certain elements. When a particular application or agent writes a message into the shared memory (namely it adds or updates an element in the database), all applications and agents having registered an observer for that particular element receive a notification about the update. This mechanism makes the database an effective communication medium in our ubiquitous AR framework.

Distributed AR applications consist of several application instances controlled by a dedicated master or *application control*. The application control maintains the global application state and distributes updates to all instances. Similarly, UbiAgents possess a single brain that controls multiple bodies embedded into individual application instances. Application instances offer only an incomplete local view of the global application state for agent bodies, while agent bodies serve only as limited proxies of the agent brain for application instances. To overcome these limitations, communication between agents and applications happens at a higher level: the application control and the agent brain exchange messages through the database, controlling actions of application instances and agent bodies.

The application control maps internal application state to public database elements. The mapping function implements the schema concept described in Section 3.3. The application control creates a transparent interface between multiple agents and applications to hide private implementation details. This interface enables already existing complex AR systems to exploit agent services without any modifications in structure and code. By employing multiple application controls, different schemas can be supported, allowing a versatile use of the application in various agent systems.

Our UbiAgent components and AR applications are built on the *Studierstube* AR framework [34], and implemented as scene graphs based on Open Inventor, a multiplatform high-level 3D graphics API. In Inventor all scene graph objects interact with one another via input and output attributes or "fields", which also provide control variables for the C++-based control logic of the scene graph objects. The brain implementation of our UbiAgents in the demo applications is also currently based on C++ code, however, dynamic scripting approaches for scene graphs such as Pivy [35] enable the dynamic uploading of procedural code, making the use of agents more flexible.

The application control marks fields relevant to the application state with a special tag, which allows constant observation of their values. A disadvantage of our scene graph approach is the lack of support for legacy applications. Interfaces to legacy applications must be implemented on a case-by-case basis.

4.2. Agent Migration

We use the Muddleware real-time XML database [36] to implement the UbiAgents' knowledge base and shared application and agent memory. Muddleware provides fast and robust access to the UbiAgent database, which has a well-defined hierarchical structure (see Figure 5c).

The Muddleware technology uses XPath-based database queries and observers. The XPath language syntax well suits the hierarchical structure of the UbiAgent database and enables the use of complex queries and observers to receive information about UbiAgent components. With XPath expressions agents can quickly identify habitats matching a set of infrastructure requirements such as hospitability, display parameters, tracking data, and application and agent attributes.

The agent brain controls a finite state machine. Each state defines a set of desired actions for agent bodies. When entering an agent state, an observer is registered to represent minimum expectations about the ideal environment for the agent bodies' actions. If the observer reports the appearance of a more suitable habitat, the agent brain instructs the currently embedded agent body to migrate to the new location.

Migration can happen in two ways: either by serialization techniques of distributed shared scene graphs [12] to create a new agent body, or by activating already existing "sleeping" agent bodies while deactivating previous ones. We also created a GUI-based UbiAgent browser to issue custom queries for debugging purposes and to trigger forced agent migrations for simulations.

5 APPLICATIONS

We created two pilot applications to illustrate our design concepts. The first application presents various migration scenarios to bridge the gap between disjoint workspaces. The second application focuses on application encapsulation and intercommunication aspects.

5.1. Character Animation Studio

Character animation projects in big studios rely on the collaborative work of several people: modelers design the mesh, animators and programmers create expressive behavior, and producers supervise all stages. The production pipeline typically requires the simultaneous use of multiple computing environments. Artists prefer to work on their personal computers, while programmers need to test characters in their target production environment such as a game console, and producers report current progress to customers in the presentation room. UbiAgent-enabled characters decrease the seam between workspaces by proactively migrating to presentation environments demanded by the current animation pipeline stage (see Figure 6).

In the design stage, a PDA is used as a tangible transfer medium for characters. The PDA is pose-tracked by a fiducial marker based on ARToolKitPlus [37], an improved version of ARToolKit, and webcams mounted on the designer PCs. If the mesh designer wants to discuss potential modifications with the animator, he/she holds the PDA in front of the webcam on the PC monitor, which indicates an intention to "pick up" the character. The character senses the PDA's spatial vicinity and "jumps over" to the handheld agent platform, which is then carried to the animator's machine. There the character migrates again to the monitor if the PDA enters a predefined "hot" area around the webcam. Changes made to the character on the animator machine are persistently stored in the UbiAgent database, therefore the next time the character is transferred back to the modeler PC, its appearance is automatically updated to reflect changes.

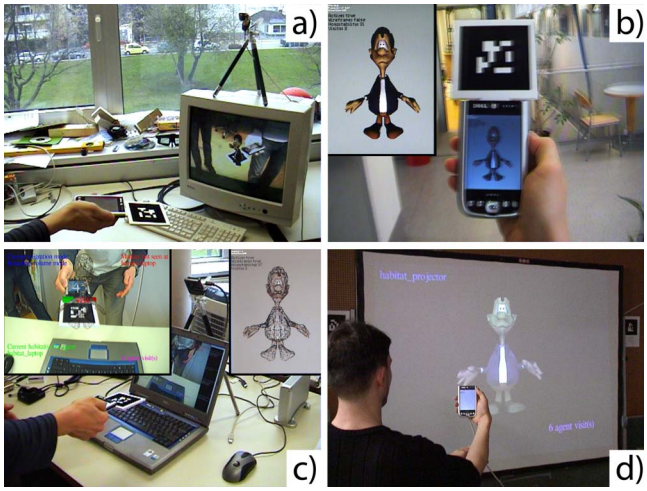

Figure 6: Improving the character animation pipeline. a) Picking up character by PDA, b) Tangible character transfer, c) Persistent agent parameter: wireframe mode preserved on PC and PDA, d) Sending character to projection screen

In the presentation stage the character is taken to the presentation room's projection screen, where an Ascension Flock of Birds magnetic tracking system is installed. By mounting a magnetic receiver on the back of the PDA and penetrating a predefined presentation area around the projection screen, the character moves from a small, private handheld display to a large public screen to show its features to a larger audience.

The continuity of the visual interface creates a spatially continuous workspace for the collaborators and thus improves productivity. To avoid discontinuities in the interface and the interaction metaphor, the characters constantly check the availability of target environments and only attempt migration if the target platforms appear to be present and "hospitable". This includes for instance the periodic checking of the handheld device's battery level as a critical resource. If the battery level is too low, the character refuses to be picked up by the PDA or escapes to the nearest available display.

To support the aforementioned scenario, character animation software packages need to be extended with UbiAgent components without modifying often proprietary internal software structure. Although our test scenario currently relies on our custom AR application framework, popular commercial animation packages can also be enhanced by special plug-ins. Firstly, each character animation application instance is encapsulated by a UbiAgent application instance object. This object runs independently from the application but continuously monitors its internal state and maps the observed state information to external parameters such as the character's rendering mode, pose, scale, current animation sequence and level-of-detail.

The external parameters are defined by a character animation attribute schema understood by a dedicated application control logic generating commands for UbiAgent-enabled animation tools. All tools using this schema (even if they internally rely on previously unknown, exotic software packages) can dynamically join the virtual UbiAgent workspace without revealing their low-level details to the character control, and thus serve as shared rendering and manipulation resources animators can exploit.

Besides the addition of the application control and instance objects, the characters themselves are represented by UbiAgent bodies controlled by the agent brain component. The agent bodies are rendered by the animation tools using the distributed character and rendering parameters retrieved from the shared database by the UbiAgent animation control. At the same time the brain component monitors the PDA's pose, and activates or deactivates agent bodies if a user penetrates a display's hotspot area and the "hospitability" parameter of the corresponding habitat is above a predefined threshold.

The modeler and animator PC, the PDA, and the presentation machine are all habitats expecting UbiAgents by running a applications to render and tweak 3D animated characters. The agent brain, the XML database, and the habitat manager run on a dedicated control PC. All components communicate via WLAN using TCP/IP and UDP messages. The handheld agent platform is installed on a Dell Axim X51 PDA with hardware accelerated graphics, which runs Daniel Wagner's Klimt and FPK libraries [36] to render an animated 3D character. The PC-based characters rely on the Cal3D character animation library [38].

5.2. Ubiquitous Technician

Technicians are a scarce resource in every research lab and company. Their never-ending to-do list is constantly extended with requests to fix malfunctioning computers, calibrate tracking systems, and maintain copy machines, all located in different

offices. The Ubiquitous Technician application provides assistance for our research lab's technician to complete various maintenance tasks on his to-do list. In this scenario we intentionally reuse elements from previous research demonstrations to test the encapsulation and communication of complex external AR applications, and to create richer content for our demo application.

The technician agent's brain creates a control loop to systematically go through the technician's to-do items (see Figure 1). As each task is located at a different place in our building, the loop first activates a modified version of the Signpost AR indoor navigation system [39] to guide the technician to the next task's location. The navigation application runs on a Sony VAIO U70 portable computer equipped with a webcam tracking fiducial markers on office walls and corridors, a UbiSense ultra-wideband (UWB) position tracker, and an inertial sensor to calculate the technician's current pose inside the building. The application displays a virtual compass suggesting the direction the technician should follow to reach the target.

This complex AR navigation system is encapsulated by a UbiAgent application controller object exposing only three attributes: current destination, current location, and a flag indicating whether the current destination has been reached. UbiAgent bodies serve as visualization tools for meaningful software components in the technician support applications. Agent bodies are activated and deactivated by a central agent brain control based on the technician's current location and the status and order of his to-do list.

After the technician arrives at the target location, the agent migrates to the AR application associated with the current maintenance task. The first job is to calibrate a cell of a UWB tracking system. An AR application [40] visualizes angle-of-arrival sensor measurements by virtual rays emanating from the physical sensors, and helps overcome problematic situations such as multipath signals caused by reflections from metal ceilings and doors. When the technician has finished the calibration procedure, he returns to the indoor navigation guide, which has already received a message with the next destination from the UbiAgent brain. Again, the technician is guided to the next task, which is the aforementioned LEGO maintenance scenario described in our previous work [1]. In this application an animated repairman assists the technician to assemble and maintain a real LEGO® Mindstorms robot.

In the Ubiquitous Technician scenario three previously independent AR applications communicate with one another using the XML database as a shared agent and application memory. New applications can be dynamically added to the to-do list by encapsulating them with an appropriate application control. The schema of the application control must contain a Signpost-compatible description of the application's location in the building, a trigger to activate the application when the technician is nearby, and a flag indicating that the application task has been completed, which instructs the agent brain to proceed to the next to-do item.

6 Discussion

Weiser [41] questions the usefulness of embodied interface agents by juxtaposing them with Ubicomp systems. He argues that assistant-like interfaces increase the seam between humans and computers, which conflicts with the fundamental goals of Ubicomp. We believe that empowering our interface agents with proactive behavior minimizes the required explicit user input to ensure correct agent operation, which makes agent presence less apparent in the interface. We also agree with Weiser's argument that some users do want personal assistants acting at their command, therefore the potential use of embodied animated agents in Ubicomp environments is justified.

User preference for agent representations spans a wide spectrum between lifelike and non-anthropomorphic embodiments. The robot maintenance application used a human-like animated character, while the calibration aid employed simple geometrical shapes to visualize application state. While non-verbal communication may increase the information bandwidth of agents, in some AR systems a simple arrow may prove more useful than a full-body animated character. A possible solution to match preferences and purposes of a wide range of users and applications may be the employment of multiple agent bodies with varying level of realism and detail. The appropriate agent body would either be explicitly selected by the user, or automatically chosen by the application matching the amount of information currently shown on the display to avoid clutters.

Another important variable in agent systems is the amount of proactivity ranging between submission and aggression. Humans are normally suspicious about systems that exclude users from the decision making loop. On the other hand, the complexity of computing systems including AR systems will soon reach a level where direct manipulation interfaces become so saturated with controllable parameters that users will have no other choice than delegating interface manipulation tasks. We also share Lieberman's point [25] that agents are rather suited for making uncritical decisions, therefore we should let agents *make a suggestion* instead of immediate actions. A typical UbiAgent example for this approach is the grace period allowing appropriate user response before agent migration.

7 Conclusion and Future Work

In this paper we proposed a framework for adaptive AR systems and discussed techniques hitherto unexplored in AR such as multi-user interface adaptation, proactive interface migration and opportunistic exploitation of dynamic resources. We argue that animated interface agents equipped with typical characteristics of Ubicomp systems enrich interaction in AR.

We created our own ontology for adaptive AR systems without the intention of completeness. However, an exhaustive ontology would be highly desirable, in particular if based on standards such as WSDL [42]. This would allow information sharing between UbiAgents and other AR and agent systems, supporting resource sharing between diverse computing systems. Another important issue for future work is to eliminate vulnerability caused by the current framework implementation based on a single central database, which makes our system prone to network and computer failures.

Acknowledgements

This project has been sponsored by the Austrian Science Fund FWF (contract No. Y193). The authors wish to thank Joseph Newman for valuable discussions on UbiAgent concepts, Daniel Wagner for his Muddleware and FPK software packages, and Gerhard Schall for his help with the Signpost system.

References

[1] István Barakonyi, Thomas Psik, and Dieter Schmalstieg. Agents That Talk and Hit Back: Animated Agents in Augmented Reality. In *Proc. of International Symposium on Mixed and Augmented Reality (ISMAR'04)*, Arlington, VA, USA, pp. 141-150, 2004.

[2] Anton Nijholt, Thomas Rist, and Kees Tuijnenbreijer. Lost in Ambient Intelligence? Extended abstract in *Proc. of Conf. on Human Factors in Comp. Systems (CHI'04) Workshop*, Vienna, Austria, pp. 1725-1726, 2004.

[3] Tomas Akenine-Möller and Eric Haines. *Real-time Rendering*, A K Peters ltd., 2nd edition, 2002.

[4] Ryan Bane and Tobias Höllerer. Interactive Tools for Virtual X-Ray Vision in Mobile Augmented Reality, In *Proc. of International Symposium on Mixed and Augmented Reality (ISMAR'04)*, Arlington, VA, USA, pp. 231-239, 2004.

[5] Steven Feiner, Blair MacIntyre, and Doree Seligmann. Knowledge-based Augmented Reality. In *Communication of the ACM*, 36 (7), pp. 52-62, 1993.

[6] Simon Julier, Marco Lanzagorta, Yohan Baillot, and Dennis Brown. Information Filtering for Mobile Augmented Reality. In *Computer Graphics and Applications*, 22 (5), pp. 12-15, 2002.

[7] Joseph Newman, Martin Wagner, Martin Bauer, Asa MacWilliams, Thomas Pintaric, Dagmar Beyer, Daniel Pustka, Franz Strasser, Dieter Schmalstieg, and Gudrun Klinker. Ubiquitous Tracking for Augmented Reality. In *Proc. of International Symposium on Mixed and Augmented Reality (ISMAR'04)*, Arlington VA, USA, pp. 192-201, 2004.

[8] Ed Kaiser, Alex Olwal, David McGee, Hrvoje Benko, Andrea Corradini, Xiaoguang Li, Phil Cohen, and Steven Feiner. Mutual Disambiguation of 3D Multimodal Interaction in Augmented and Virtual Reality. In *Proc. of International Conference on Multimodal Interfaces (ICMI'03)*, Vancouver, Canada, pp. 12-19, 2003.

[9] Hrvoje Benko, Edward W. Ishak, and Steven Feiner. Cross-Dimensional Gestural Interaction Techniques for Hybrid Imrnersive Environments. In *Proc. of Virtual Reality Conference (VR'05),* Bonn, Germany, pp. 209-216, 2005.

[10] José Pascual Molina Massó, Jean Vanderdonckt, and Pascual González López. Direct Manipulation of User Interfaces for Migration. In *Proc. of International Conference on Intelligent User Interfaces (IUI'06)*, Sydney, Australia, pp. 140-147, 2006.

[11] Asa MacWilliams, Christian Sandor, Martin Wagner, Martin Bauer, Gudrun Klinker, and Bernd Brügge. Herding Sheep: Live System Development for Distributed Augmented Reality. In *Proc. of International Symposium on Mixed and Augmented Reality (ISMAR'03)*, Tokyo, Japan, pp. 123-132, 2003.

[12] Dieter Schmalstieg, Gerhard Reitmayr, and Gerd Hesina. Distributed Applications for Collaborative Three-Dimensional Workspaces. In *Presence: Teleoperators and Virtual Environments*, 12 (1), pp. 52-67, 2003.

[13] Jun Rekimoto and Masanori Saitoh. Augmented Surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments. In *Proc. of Conference on Human Factors in Computing Systems (CHI'99)*, Pittsburgh, PA, USA, pp. 378-385, 1999.

[14] Andreas Butz, Tobias Höllerer, Steven Feiner, Blair MacIntyre, and Clifford Beshers. Enveloping Users and Computers in a Collaborative 3D Augmented Reality. In *Proc. of International Workshop on Augmented Reality (IWAR'99)*, San Francisco, CA, USA, pp. 35-44, 1999.

[15] Ben Shneiderman and Pattie Maes. Direct Manipulation vs. Interface Agents. Excerpts from Debates at IUI 97 and CHI 97. In *ACM Interactions*, 4 (6), pp. 42-61, 1997.

[16] Pattie Maes, Trevor Darrell, Bruce Blumberg and Alex Pentland. The ALIVE System: Wireless, Full-body Interaction with Autonomous Agents. In *ACM Multimedia Systems*, 5 (2), pp.105-112, 1997.

[17] Mahoro Anabuki, Hiroyuki Kakuta, Hiroyuki Yamamoto, and Hideyuki Tamura. Welbo: An Embodied Conversational Agent Living in Mixed Reality Space, In *Proc. of Conference on Human Factors in Computing Systems (CHI'00), Extended Abstracts*, The Hague, The Netherlands, pp. 10-11, 2000.

[18] Blair MacIntyre, Jay D. Bolter, Jeannie Vaughan, Brendan Hannigan, Emmanuel Moreno, Markus Haas, and Maribeth Gandy. Three Angry Men: Dramatizing Point-of-View using Augmented Reality. In *Proc. of SIGGRAPH 2002 Technical Sketches*, San Antonio, TX, 2002.

[19] Adrian Cheok, Wang Weihua, Xubo Yang, Simon Prince, Fong S. Wan, Mark Billinghurst, and Hirokazu Kato. Interactive Theatre Experience in Embodied and Wearable Mixed Reality Space. *In Proc. of International Symposium on Mixed and Augmented Reality (ISMAR'02)*, Darmstadt, Germany, 2002.

[20] Marc Cavazza, Olivier Martin, Fred Charles, Steven J. Mead, and Xavier Marichal. Interacting with Virtual Agents in Mixed Reality Interactive Storytelling. In *Proc. of Intelligent Virtual Agents*, Kloster Irsee, Germany, 2003.

[21] Selim Balcisoy, Marcelo Kallmann, Rémy Torre, Pascal Fua, and Daniel Thalmann. Interaction Techniques with Virtual Humans in Mixed Environments. In *Proc. of International Symposium on Mixed and Augmented Reality (ISMAR'01)*, Tokyo, Japan, 2001.

[22] Luca Vacchetti, Vincent Lepetit, George Papagiannakis, Michal Ponder, and Pascal Fua. Stable Real-time Interaction between Virtual Humans and Real Scenes. In *Proc. of International Conference on 3D Digital Imaging and Modeling (3DIM'03)*, Banff, AL, Canada, pp. 449-457, 2003.

[23] Brenda Laurel. Interface Agents: Metaphors with Character. In *The Art of Human-Computer Interface Design*, edited by B. Laurel, Reading, MA, USA, Addison-Wesley, 1990.

[24] Stan Franklin and Art Graesser. Is it an Agent or just a Program? A Taxonomy for Autonomous Agents. In *Agent Theories, Architectures and Languages*, Springer Verlag, Berlin, Germany, pp. 21-95, 1996.

[25] Henry Lieberman. Autonomous Interface Agents. In *Proc. of Conf. on Comp. and Human Interface (CHI'97)*, Atlanta, GA, USA, pp. 67-74, 1997.

[26] David Kotz and Robert S. Gray. Mobile Agents and the Future of the Internet, In *SIGOPS Operating Systems Review*, 33 (3), pp. 7-13, 1999.

[27] Kenji Mase, Yasuyuki Sumi, and Rieko Kadobayashi. The Weaved Reality: What Context-aware Interface Agents Bring About. In *Proc. of Asian Conference on Computer Vision (ACCV'00)*, Taipei, 2000.

[28] Daniel Wagner, Mark Billinghurst, and Dieter Schmalstieg. How Real Should Virtual Characters Be? To appear in *Proc. of Conference on Advances in Computer Entertainment Technology (ACE'06)*, Los Angeles, CA, USA, 2006.

[29] Michael Kruppa and Antonio Krueger. Concepts for a Combined Use of Personal Digital Assistants and Large Remote Displays. In *Proc. of Simulation and Visualization (SIMVIS'03)*, Magdeburg, Germany, pp. 349-361, 2003.

[30] Bill Tomlinson, Man Lok Yau, and Eric Baumer. Embodied Mobile Agents. To appear in *Proc. of International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'06)*, Hakodate, Japan, 2006.

[31] Brian R. Duffy, Gregory M.P. O'Hare, Alan N. Martin, John F. Bradley, and Bianca Schön. Agent Chameleons: Agent Minds and Bodies. In *Proc. of International Conf. on Computer Animation and Social Agents (CASA'03)*, New Brunswick, NJ, USA, 2003.

[32] Michael E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, USA, 1987.

[33] Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. The Belief-Desire-Intention Model of Agency. In *Proc. Of Intern. Workshop on Intelligent Agents*, Heidelberg, Germany, pp. 1-10, 1999.

[34] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavári, Miguel Encarnação, Michael Gervautz, and Werner Purgathofer. The Studierstube Augmented Reality Project. In *PRESENCE - Teleoperators and Virtual Environments*, MIT Press, 2002.

[35] Pivy website: http://pivy.tammura.at/

[36] Handheld AR libraries website (Muddleware, Klimt, FPK): http://studierstube.icg.tu-graz.ac.at/handheld_ar/

[37] ARToolKitPlus website: http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php

[38] Cal3D website: http://cal3d.sourceforge.net/

[39] Michael Kalkusch, Thomas Lidy, Michael Knapp, Gerhard Reitmayr, Hannes Kaufmann, and Dieter Schmalstieg. Structured Visual Markers for Indoor Pathfinding, In *Proc. of the First International Workshop on ARToolKit*, 2002.

[40] Joseph Newman, Gerhard Schall, István Barakonyi, Andreas Schürzinger, and Dieter Schmalstieg. Wide Area Tracking Tools for Augmented Reality. In *Advances in Pervasive Computing 2006*, Vol. 207, Austrian Computer Society, Vienna, 2006.

[41] Mark Weiser. Does Ubiquitous Computing Need Interface Agents? *MIT Media Lab Symposium on Interface Agents*, Cambridge, MA, USA, 1992.

[42] WSDL website: http://www.w3.org/TR/wsdl