An Approach for Developing Ubiquitous Augmented Reality Systems

Bernd Bruegge
Asa MacWilliams
Technical University of Munich

Asim Smailagic
Carnegie Mellon University

1. Introduction

Augmented reality is an interaction technology that provides users with spatially related information in the real world in real time [1]. An augmented reality system tracks the position and orientation of objects, for example the user's head, and superimposes virtual objects into the user's field of view. Augmented systems support multiple human-computer interaction techniques for input (multi-modality) and output (multi-media). Ubiquitous computing aims to provide a "calm" interface to computers, making them invisible, yet omnipresent [2]. Users should be able to use computing technology in their environment without having to think about it as such. For this, devices must provide their services autonomously and shield the user from configuration tasks. Additionally, other users can bring along devices which themselves provide new services. Ubiquitous computing is a broader field than augmented reality, and includes many different ways of augmenting the users' environment, with augmented reality as one possible technology. As the computing and interaction hardware required for augmented reality becomes smaller and more ubiquitously available, augmented reality and ubiquitous computing can converge, creating systems that are ubiquitously available and allow interaction in the style of augmented reality.

In ubiquitous augmented reality applications, users are involved with tasks in the real world, and they cannot solve these tasks by sitting in front of a desktop computer. This includes mobility: users move around in a fairly large, inhomogeneous area. At the same time, users wish to access or modify information associated with their current real-world situation, which is not otherwise immediately available.

Building ubiquitous augmented reality systems presents a challenge [3], because the requirements are still ill-defined, as appropriate interaction metaphors are still being researched and users' preferences change.

To deal with this problem, this paper proposes a new prototyping style called extreme modeling. In extreme modeling, users and developers address ill-defined requirements by changing the system's behavior during development as well as while it is running, facilitating the creation of new applications. We describe three prototype systems that have been built with this methodology.

2. Extreme Modeling

In extreme modeling, the system is not only deployed incrementally, but also be developed incrementally as well. Thus, parts of the system will be deployed — and tested by the end user — before others are even developed. In extreme modeling, developers in collaboration with end users incrementally improve a system while users use it and provide feedback. This follows the idea of shorter development cycles, as, for example, in extreme programming [4] or Scrum [5]. Two techniques that can be used in the process are described: jam sessions, where users and developers cooperate synchronously, and continuous extension, where they cooperate asynchronously.

2.1 Jam Sessions

A jam session is a synchronous group activity of users and developers, cooperating in improving a running system, which is deployed within one room, or a small site consisting of several adjacent rooms. A jam session takes from a few hours up to a whole workday. In developing a system, users and developers may wish to have several jam sessions consecutively. The basic unit of work in a jam session is the feedback–develop cycle. When the end user finds a fault or suggests an improvement in a component, a developer changes the component, taking it off-line briefly, and restarting it while the rest of the system remains running. Or a developer implements several alternative versions or configurations of a component, and the user can test and chose among them.

In a jam session, several of these develop–feedback cycles take place simultaneously, each involving at least one user and one developer. While one part of the system changes, the rest continues to run, and thus different parts of the system are improved simultaneously. A jam session does not have to be centrally

coordinated; it follows the idea of a jam session in music, where participants take the lead in turn. At times, one group of developers and users will have to defer a change to the system, because it would interfere with a test being performed by another group. This kind of group coordination requires a certain level of experience from developers and users. A jam session also requires support from the architecture and run-time infrastructure for ubiquitous augmented reality systems: in particular, the software architecture must allow components to be exchanged while the system is running.

Jam sessions are particularly suitable for building prototypes of ubiquitous augmented reality applications as well as evaluating and improving their usability. A jam session is less suitable for the development of new algorithms within a specific component, or for performance tuning.

2.2 Continuous Extension

In contrast to jam sessions, continuous extension assumes that developers and users cooperate asynchronously, and that the system supports their collaboration. Continuous extension assumes that a system has been - at least partially - deployed and is in use. When a user discovers a problem or an opportunity for improvement, he uses a feedback tool, which is part of the system, and records a wish for improvement. The system stores this wish, together with the current user and system context—such as the state of services the user is using, the current graph of services, and the user's location. The Kiva, described in [6], is an example of such a system.

The system can also gather usage information automatically, by creating profiles of which services get used in which circumstances, whether the performance is adequate, or how long it takes the user to complete a task. Using the information that the system has gathered, developers implement missing functionality, improve existing services, and fix bugs.

This can happen off-line in a development laboratory. To test new and experimental features, they can even use jam sessions again, either with the same users who requested new functionality, or with selected pilot users. The new versions are then deployed to the hardware in the environment, and onto the users' mobile hardware.

If the components are backward-compatible, this can happen in the background, without users noticing. Once the new components are dynamically deployed, users can take advantage of the improved functionality, and a new feedback–development cycle can begin.

As with jam sessions, many of these cycles are active at any given time, as users can provide feedback asynchronously. When context changes or the user gives the system an appropriate command, it can adapt itself immediately—assuming it "knows" how. Assuming a speech recognition interface, the user can say, "show me an asana for improving my upper body strength", and a wall-sized display in the environment is reconfigured to show the asana. On the other hand, the system may not be able to react immediately when the required functionality is missing. In this case, the system reacts by saying "sorry, I cannot do that yet", record the user's wish, and a developer implements the missing functionality. When the user tries the command again the next day, the system can fulfill it.

The concept of end user programming allows users to reprogram their ubiquitous computing systems with a simple visual interface. A simple example is a rule-based system: users define rules telling the system what to do in which circumstances. By contrast, in continuous extension, users do not program directly, but record requests for developers. This is based on the assumption that most users of a ubiquitous augmented reality system are not programmers. Of course, continuous extension can be combined with end user programming; users can define simple rules and add simple functionality themselves, getting immediate results; and professional developers handle the more complex programming tasks.

3. Case Studies: Skill-Coaching Systems

Teaching a skill involves four basic steps: Preparation, Explanation, Demonstration and Practice. The explanation introduces the subject and talks about its usefulness and applicability. During demonstration the teacher shows each step so that the learners can easily follow and gain confidence in their own ability to acquire this skill. During practice the learners try out the skill under the teacher's guidance

When teaching a skill, the focus is on the learners; the teacher is trying to help *them* gain this skill and be able to use it with a sense of comfort and confidence. For complicated skills such as yoga, playing piano or conducting, practicing the skill under the teacher's guidance is crucial, because learning these techniques from a book, tape or video is severely limited. The (real) teacher cannot be present all time during practicing. We argue that ubiquitous augmented reality can be used to replace a teacher periodically in practice sessions. In this case the interaction between computer and learner during the practice session is especially important.

We argue that the interaction should have the following properties:
• **Adaptivity**: If the learner is not familiar with the skill, the system must adapt to the learner. For example, the system should be able to first check on accuracy and then for speed, if necessary.
• **Noninterference**: The system should not interfere when learners try to do it on their own. In particular, the system should not interrupt their efforts unless they bog down or go off on the wrong track.
• **Support for trial-and-error**: The learners should be allowed to make mistakes if this will help them learn. That is, the system needs to distinguish between a moment where it should interfere and when not.
• **Feedback:** In the case the system needs to point at mistakes, it should do it tactfully. In particular, the system needs to provide multi-modal output to select from a variety of reactions depending on the learner situation and context. For example, when a yoga student violates a position that requires symmetry, the system should be able to select from feedback in form of a spoken instruction or different types of music indicating the severity of the violation.
• **Annotations:** Encourage the learners by making remarks on their progress, pointing out the completion of each step, and remarking on the steps they have done well.
• **Learning**: The system should be able to learn from the interaction with the learner and adjust the next interaction based on this knowledge.

Rapid prototyping has successfully been used to develop GUI based systems: The developer shows a graphical interface to end user or domain expert, who makes comments that may lead to the revision of the user interface. Extreme modeling goes beyond rapid prototyping. The validation of the system is no longer completely done during development. The development of the system continues during the use of the system, even to the point that the user has to interact with the system to revise some of its behavior, which can be seen as a change of the requirements. That is, two types of interaction must be supported: 1) Interactions to practice the skills with the learner and 2) interactions that allow the domain expert and/or end user to change the system at runtime to improve its usability. The same mechanisms can be used for both types of interactions. This requires that the underlying software architecture must peer-to-peer and not client-server based: The system has to act as a true peer, being able to initiate an interaction or reacting to an unplanned interaction initiated by the user.

3.1 Personal Yoga Instructor

We developed a context-aware application for learning Indian Yoga, in particular Hatha Yoga. Yoga is poised to become a major theme for the wellness market of the near-to mid-term future. A student learning Yoga attends a weekly Yoga class and doing exercises at home between the classes. In the Yoga class, the qualified Yoga teacher provides advice, but at home, nobody guides the student to perform the exercises correctly. We built a "Personal Yoga Instructor" (PYI). The PYI, shown in Figure 1, gives advice on planning and selecting appropriate daily and weekly Yoga exercises, provides direct feedback on achievements through non-intrusive interaction, and alerts students to damaging postures with Yoga poses. The PYI is an example of a new class of applications that use smart sensors and novel interaction modalities emerging in the next decade that move beyond mobile peer-to-peer voice and text message communication.

3.2 Interactive Conducting

We are going to use accelerometers on a small wearable device and machine learning algorithms to recognize the arm movements of a symphony orchestral conductor. This will be an effective mechanism to support the training and practicing of conductors. We are planning to perform testing with a symphony orchestra later this year. We are also aiming to use gesture recognition and compare recognition accuracy. The system will provide feedback to conductors to improve their performance.

3.3 Wellness Coach

The wearable Personal Wellness Coach system [7], based on our machine learning algorithms described in [8], and wearable computing [9], supports a range of activities including health data collection, interpretation, feedback and self-monitoring. It is shown in Figure 2. We implemented a wearable computing platform consisting of an array of sensors in the form of an armband, and application software to motivate users to reach fitness goals and prevent harm in a real-time environment. Evaluation and testing were done on 65 users in four groups: students, health club members, ROTC members and Baby Boomer-age individuals. Comparing the Personal Wellness Coach against an existing exercise product, our results indicate that this system is an effective tool that is less intrusive and in some cases can perform like a personal trainer and help improve user efficiency.

Figure 1. Personal Yoga Instructor

Figure 2. Personal Wellness Coach

4. Discussion Points for the Workshop

- For which types of systems is extreme modeling applicable?
- How do we interact with systems whose behavior is not reproducible?

5. Bibliography

[1] Azuma, R. T., "A Survey of Augmented Reality," *Presence*, Vol. 6, No. 4, August 1997, pp. 355–385.

[2] Weiser, M., "Hot Topics: Ubiquitous Computing," *IEEE Computer*, Oct. 1993.

[3] Asa MacWilliams, A Decentralized Adaptive Architecture for Ubiquitous Augmented Reality Systems, Dissertation, Technical University Munich, 2005.

[4] Beck, K., *eXtreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[5] Schwaber, K. and Beedle, M., *Agile Software Development with Scrum*, Prentice Hall, 2002.

[6] Finger, S., Gelman, D., Fay, A., Szczerban, M, Smailagic, S., Siewiorek, D.P., "Supporting Collaborative Learning in Engineering Design," International Journal of Expert Systems and Applications, to appear, 2006.

[7] Asselin, R., Ortiz, G., Pui, J., Smailagic, A., Kissling, C., "Implementation and Evaluation of the Personal Wellness Coach," Proc. IEEE International Workshop on Smart Appliances and Wearable Computing, June 2005, Columbus, OH

[8] Krause, A., Smailagic, A., Siewiorek, D.P., "Context-Aware Mobile Computing: Learning Context-Dependent Personal Preferences from a Wearable Sensor Array," IEEE Transactions on Mobile Computing, Vol. 5, No. 2, 2006, pp. 113-127.

[9] Smailagic, A., "Wearable Computers: A New Paradigm in Computer Systems and Their Applications," IEEE Transactions on Computing, Vol. 52, No. 8, 2003, p. 977