AN ARCHITECTURE FOR PRESENTING AUDITORY AWARENESS INFORMATION IN PERVASIVE COMPUTING ENVIRONMENTS

Anssi Kainulainen, Markku Turunen, and Jaakko Hakulinen

University of Tampere
Department of Computer Sciences
Tampere Unit for Computer-Human Interaction
Speech-based and Pervasive Interaction Group
Kanslerinrinne 1, FIN-33014 University of Tampere, Finland
{Anssi.Kainulainen, Markku.Turunen, Jaakko.Hakulinen}@cs.uta.fi

ABSTRACT

In this paper we present how awareness can be supported in pervasive computing environments through auditory information. We introduce an application which uses soundscapes to support people's awareness of each other's presence in an office environment. We describe several techniques for construction and control of such soundscapes. Finally, we present an architecture for designing and controlling soundscapes. The architecture is based on managers, agents, evaluators, a blackboard information storage, and a control language, it emphasizes reusability and extensibility, and it is built upon a common system framework.

1. INTRODUCTION

Group awareness, or awareness of co-workers, status of the tasks and activities in a work group, is important to co-work and social life of individuals. Supporting awareness in pervasive computing environments benefits large target audiences in wide variety of tasks, social situations and physical environments. The users can interact with each other normally, and collaborate in normal daily tasks, since the awareness support is available in the environment itself. The awareness support can be done in a calm and unobtrusive manner, exploiting the peripheral sensing capabilities of humans.

Pervasive computing environments produce their own challenges to awareness support of work groups. Gathering information regarding the users, their work, tasks and social situations is a much more complex situation than in traditional Computer Supported Cooperative Work (CSCW), which focuses on systems based on personal computers and desktop applications. A wider variety of information is available, and the scale of topics and domain information is much larger. Information gathering and reasoning techniques are also complicated. Information sources can typically be, for example, an array of recognition results from speech and speaker recognizers, positioning services, and other software for gathering information. Handling that complexity is a challenge in itself. Such fusion of data, or context information, is often called context awareness. Our approach to awareness support is to present cues for people to act, and keeping humans in the loop, instead of "automating actions", which Bellotti and Edwards [1] warn against.

In this paper we introduce an application to support people's awareness of each other's presence in an office environment. We use calm and continuous soundscapes such as ones based on sounds of birds singing and people walking [2] to convey the information while not burdening the listener's cognitive load. The awareness information application is part of a larger pervasive computing system [3], which aids people at the office in many forms, e.g., by providing indoor guidance, spoken messages, and other relevant services.

In order to construct efficient soundscapes to present awareness information, a software architecture with sufficient capabilities for manipulating and controlling sounds and compositions is needed. Particularly, there is a need to ensure the consistency and context sensitivity of the dynamic presentation. This requires special techniques for transitions, timing, mixing, adaptivity to outside information sources and for handling the complexity of controlling such a compound presentation. We present a conceptual model and a concrete software architecture with a basic set of components that enable the application designers and soundscape composers to construct efficient auditory awareness applications. The components consist of four categories of agents, a central information storage, and sound engines, all controlled with a markup and scripting language. The architecture is based on our experiences in constructing [3] and evaluating [4] auditory pervasive computing applications. Technology-wise, the model extends our generic agent architecture [5].

In order to understand the importance of awareness support, we start by briefly describing the psychological mechanisms of peripheral awareness, and how they can be utilized. Then we present an application which supports awareness through calm soundscapes. After that, we take a closer look at different challenges in producing such soundscapes: the mapping of information to presentation, maintaining the consistency of a presentation, and otherwise controlling them. Then we describe an architecture designed to cater for all these requirements. We conclude with discussion and plans for future work.

2. PERIPHERAL AWARENESS

In this paper, awareness of co-workers, status of the tasks and activities in a work group is called group awareness. Dourish and Bellotti [6] define awareness as "understanding of the activities of others, which provides a context for your own activity." An individual's social effect on others can be seen as a sum of social bonds, physical proximity and temporal immediacy. Thus, the role of awareness, even just of presence of others, is important to co-work and social life of individuals.

Nonetheless, bringing new output devices to an environment and adding the amount of information available to people is potentially harmful, since that easily adds to an information overload which might lead to stress and decrease in productivity. Therefore, a major challenge in supporting awareness is offering information on an unobtrusive level of attention.

In a cognitive sense, awareness is closely tied to attentiveness. Attentiveness shifts between categories from preattention and inattention to divided attention and focused attention [7]. The shift is involuntary or voluntary, depending on the situation, which means that awareness support applications have to be unobtrusive in the sense that they do not demand attention unnecessarily with interruptions or roughness. The level of obtrusiveness has to also match the social and task situation. Presentations have to be consistent and continuous.

Pervasive computing applications utilizing the peripheral sensing capabilities of humans can convey information of which people are aware of, but have not focused their attention to [8]. Inattention and divided attention are used in calm ambient presentations. Inattention does not burden a person's cognitive capacity. In this paper we study how this can be achieved with audio applications which are designed to convey awareness information in unburdening ways. Next we will describe an application which uses calm soundscapes as the method of presenting information.

3. AN AUDITORY AWARENESS SUPPORT APPLICATION

In order to support group awareness, as presented in Section 2, we have implemented an application which aims to keep people aware of each other's presence in the office environment. The application is part of a larger pervasive computing system that helps people in their everyday tasks in office settings [3]. The system gathers presence information from various sources, for example by tracking the frequency people use their computers. The auditory awareness information is presented through loudspeakers embedded in the office environment.

Figure 1 illustrates generic properties of the auditory awareness support application. The first phase is to collect data from various sources, such as implicit activity information from movement sensors, and explicit information from personal computers (e.g., calendar applications). The second phase is the refinement of the information, for example by using rule-based or statistical reasoning to make higher level abstractions. In the third phase the mapping between the awareness information and available auditory presentation methods is made. In the fourth phase the resulting auditory presentation is sent to the audio engine to be played out.

In our awareness support application, information is presented in the form of soundscapes that consist of thematically similar sounds, such as sounds of birds singing. Traditionally soundscapes mean the entire sonic environment as it is sensed and interpreted by its listeners. We use the term narrowly to mean the sounds created by the described application, the design of which is nevertheless attached to the physical environment and its sounds. The activity and presence of each person of the work group is depicted by the singing of a bird [BIRDSCAPE_01.OGG]. In early mornings and late evenings when only a few people are present, the soundscape reminds a quiet lakeside in the wilderness with just a few bird calls echoing over the water. During active moments there are more birds creating a more active atmosphere. In addition, we have experimented with several different kinds of sounds, such as sounds of walking [2], and means of constructing soundscapes and compositions out of them.

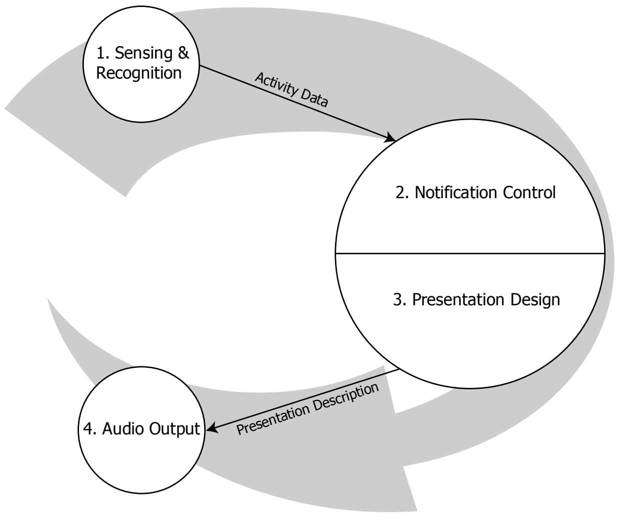

Figure 1. *A generic model for auditory awareness support applications.*

The application collects presence and activity information by tracking how often people use their computers. Activity within a certain timeframe (5 min) is interpreted to be recent enough, and the location of the activity is interpreted as the location of the person. This rather simple model is based on our iterative design experiences. The rationale is further explained in Section 4.1.

The application mixes sounds dynamically together according to certain constraints. To avoid congestion at the busiest moments, only a certain amount of sounds (3) are allowed to play simultaneously. Sounds are mixed to partially overlap, and a new sound can be introduced to the soundscape only when the previous sounds are at a suitably calm point, e.g., during pauses or low-energy points in bird calls. In our installation, people's offices are all in a row along the same hallway, the seating order is used as the order of sounds. These rules make the soundscape seem like a moving glance around the environment.

While implementing the application we came across many challenges. There is a strong need for tools of composition and dynamic control. The application must have an easy to use interface for iterative design and experimenting with different kinds of compositions. On the other hand, the application must have an efficient and flexible protocol for communicating with the information sources and other applications. These two viewpoints must come together in a dynamic and interactive manner. We tackled these challenges by including easy to use markup- and scripting languages for the composers to define the soundscape presentations. At the same time, the languages allow a clearly defined format for application developers to implement applications. The functionality of the components is also configurable in the same manner, which allows quick experimenting and reuse. The markup- and scripting languages are described in Section 5.3. In the following sections we will present requirements and technical challenges set for sound production and presentation in more detail.

4. ELEMENTS OF AUDITORY SOUNDSCAPES

While developing the application described in the previous chapter, we learned that the three most important areas of auditory soundscape presentation development are mapping of information, handling of continuity and consistency, and control of compositions. Information can be mapped to concrete presentations in many ways, which are not trivial and sometimes require musical knowledge and skills. The continuity and consistency of a presentation affect its calmness and unobtrusiveness. There are many methods of defining and controlling whole compositions. In the next sections we will present our experiences, design rationale, and some general findings. Special attention is paid to adaptivity and interactivity of the audio presentation.

4.1. Mapping Concepts to Presentations

Generally, soundscapes are created by combining multiple individual sounds together, defining rules and variation for their qualities, behavior and interaction. When the presentation is well designed, the listeners can easily understand the changes in the soundscape, the objects moving in it and interacting with each other, as a whole. In order to present meaningful information using audio we need to define the mapping between the information and its auditory presentation. The mapping can be:

- a direct connection between a variable and a sound, where the change of the value of a variable causes a change in the sound or it's qualities directly, as a continuous stream or as finite auditory icons;
- static, binary or dynamic, where static sounds can present atmospheric, default and unchanging information, binary sounds can present the presence of a person or other true/false information, and dynamic sounds can present information on processes and interaction;
- hierarchically structured, where, for example, sounds can be related to multiple categories at the same time determined by their melodies, tones and harmonies; and
- complete compositions or virtual constructs of objects, where the elements are viewed as individual objects with rules and patterns guiding their behavior and their interaction.

Many of these design challenges have been addressed in previous work like the Khronika System [9] and ENO [10]. An additional design challenge is that of the uncertainty of context information, as described by Korpipää [11].

After a soundscape has been defined, it can be rendered much in the same way as three dimensional graphics; continuously in real-time. The interaction of individual elements generates the richness of the whole presentation. All this requires ways to bind different types of information to sound elements and compositional structures in different levels of abstraction, which can be dynamically edited in real time depending on input according to rules which dictate their behavior.

4.2. Timing, Transitions, Consistency and Continuity

A sound presentation can be finite, or continuous and open-ended. Finite presentations are easier to define and construct, since they can be pre-processed before the actual presentation. Continuous presentations are harder to define and control, since their definitions change constantly while the presentations have to be kept running without pauses or breaks.

In order to create consistent and continuous presentations, timing and synchronization of sound elements is important. New sound elements and changes in the previous elements have to be timed correctly to the already playing presentation, its elements and musical passages. Changes in the information cause changes in the presentation, and these changing points, or transitions, have to be smooth and natural. The timing and handling of transitions requires a notation and control system.

A transition can be just silence between two consecutively played elements or their direct splicing, a cross-fade or synchronized overlap of common rhythms, use effects to ease the transition, or in the best case be completely seamless. A seamless transition is difficult to achieve even in musical terms, and it is even harder to automate technically. In the most difficult case, music and the system that produces it have to be ready to make the transition at any moment. The bigger the change, the more difficult it is to achieve seamlessly.

Technology-wise, transitions can be carried out by jumping in to previously marked points of the predefined composition, either within the same or to a different presentation. Transitions can also be handled by making a layered presentation. A layered transition means adding and removing sounds or tracks from the composition in layers. A transition matrix can also be used to handle the transition between every possible track combination with especially composed transition-tracks. It is also possible to track the harmonies of a piece of music [12] so that new instrument layers can begin in harmony with the previous ones. Chord maps [13] can be used to define even complicated rules after which certain chords can move into others.

Sometimes compositional elements start to repeat themselves, which can become irritating. Alternative sound elements can be randomized according to certain rules. Generative music [14] is specialized in randomness and indeterminacy.

The system and notation has to be extensible, since there is no way to predict all possible ways timing and transitioning can be performed.

4.3. Controlling Sound Presentation Compositions

The larger the amount of information to be presented is, the more demanding is the task of controlling the sound presentation during application run-time. Because of the growing complexity, it is also useful to separate application design between programmers and composers. A common, multi-layered abstraction, as mentioned in section 4.1, can support making different versions of a presentation for different use contexts, even when the information stays otherwise same. Different hardware platforms, e.g. PCs and mobile phones, can be used to render the same presentation in this way. The software component producing and controlling the sound presentation has to be flexible and versatile, and it has to offer a clear tool for designing presentations.

A soundscape can react to changes in the information to be presented directly in real-time, with a delay or even by anticipating events. Direct reaction means real-time changes in the presentation. Real-time reactivity means concurrent operation of independent components. A delayed reaction means that the change is timed for the next moment it is aesthetically or psychologically appropriate. Anticipation means preparing to previously arranged and known chains of events or statistically probable situations. By anticipating events the transition to a change in the sound presentation can be started before the change actually takes place.

One of the major challenges is to define how the software component which controls the sound presentation communicates with the rest of the system, what elements of the presentation can be changed, and how the system triggers these changes. Computer games [15; 16; 17; 18] are a good example for these challenges, since they have used different kinds of triggers in setting off changes in their sound presentations for a long time already. Similar triggering techniques can be employed to react also to real world situations. For this purpose, the sound presentation software component needs an open interface and a way to communicate back to different external information sources.

Control can also be achieved by using events and scripting to create a link between soundscapes and the underlying information sources. Scripting in this context means a way of building complex combinations and events from single rules and commands with a simple annotation mechanism or language. Technology-wise, scripting requires a language general enough, so that other soundscape control methods and components can be used, and events can be defined in a clear and easy way. This kind of scripting can be thought to be a parallel and somewhat overlapping way of defining a sound presentation to traditional musical composition. There are many kinds of scripting languages and notations, ranging from concurrent audio programming languages like ChucK [19] to synthesis markup languages like SSML [20].

5. AN ARCHITECTURE FOR DESIGNING, PRODUCING AND CONTROLLING SOUNDSCAPES

While developing our application, we faced the same problems as Dey et al. [21], i.e., a badly understood notion of context, a lack of common models and methodology, and a lack of tools for supporting application development. Just as they, we saw that providing means for more systematic application development are the next step in facing these challenges.

We introduce an architecture for producing and controlling soundscapes for pervasive computing applications. It implements the generic model presented in Chapter 3. The architecture takes into account the special needs of pervasive computing environments, where the target audience moves around in space and from social situation to another, and where work tasks and technological capabilities of the environment change. In addition, different users, such as users with special needs [5], are taken into account. For example, different versions of the same presentation can be easily created to suit the diminished capabilities of hearing impaired. The general structure of the architecture is illustrated in Figure 2.

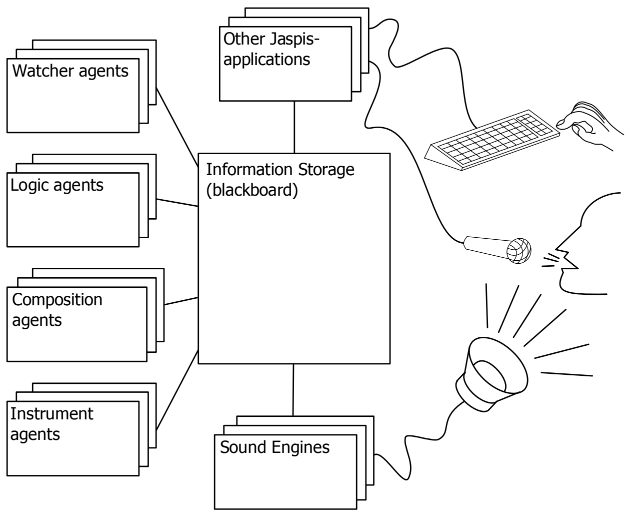

Figure 2. *The architecture consists of four groups of agents, a sound engine, and other components dividing the tasks between them.*

Technology-wise, the architecture is built on top of the Jaspis speech and pervasive computing application framework [5]. Since the architectural solutions presented here are built on top of the common underlying framework, the functionality is available also to other applications built on top of Jaspis. The structure and functionality is largely object-oriented, and it follows the agent-evaluator-manager –principle. Managers form the basic blocks and handle the coordination of the applications: information gathering management, logic management, composition management, instrument management and such. The production and control of the sound presentation is decentralized to subtasks of different agents. The agents are compact, highly specialized and numerous software components, which enables easy reuse and extension of functionality. They handle small tasks and responsibilities, and are dynamically chosen to suit the task at hand. Agents both inherit functionality and take advantage of configurability through scripting. Evaluators choose which agents to use at a specific moment, and thus handle the system level adaptation. This adaptivity is important in handling the ever-changing presentations as described in Section 4.2.

Awareness information and its auditory representation are stored in a shared system knowledge base, or Information Storage, on different abstraction levels. This kind of a blackboard is similar to Hearsay-II [22], which describes a classical way to present information in different levels of abstraction. All of the agents can modify the presentation according to their specifications and the situation. The freedom of information flow is ensured by the central Information Storage, where information is available and modifiable to all agents.

The presentation created and controlled by the agents is produced by sound engines which parse the commands and requests given by the agents. The communication works both ways, so events in the presentation can trigger further activity of the agents, and thus even of other applications through the common architecture.

The architecture supports distribution, which means that it is possible to run each agent and engine on separate platforms in different physical locations [23]. This answers to an essential requirement in ubiquitous computing environments, where hardware environments are distributed and diverse.

As described earlier (Figure 1), data can be acquired through public information providers, other applications in the same framework setup, or ones especially designed for the purpose of the awareness presentation application. Manipulation and refinement of the raw data is a separate function from the construction of the presentation. The components handling presentation of information are services which can share their resources to many applications. The same is true for information gathering components.

Next we will describe each of the main components with examples. First we will present each of the agent types and their responsibilities. Then we will describe the sound engine and its interface. We will conclude with the scripting language.

5.1. Soundscape Control Agents

The design, creation and control of soundscapes are done by agents. The combination of many agents enables us to build large applications and help keep the control easy to handle and modify and extend. Agents with atomic tasks can be used by other agents for combined tasks, and so on. Agents can also inherit functionality in normal object oriented programming ways. This distribution is important for reusability and extensibility. Agents can be roughly grouped into four categories:

- watcher agents, which convey information to and from outside sources, sensor inputs and such,
- logic agents, which state how the perceived events affect the presentation and composition in a conceptual level,
- composition agents, which design the composition and guide the instrument agents, and
- instrument agents, which use and manipulate the low level elements of the presentation description and define and create the final and concrete version which is sent to the sound engine to be rendered.

Defining the boundaries of each agent category depends on the case, and the categorization is more of a default suggestion to help the design of the presentation and the application. Nevertheless, the division is important from the viewpoints of concurrency of tasks, iterative development and the specialized skill-areas of application designers and composers. In the following, we will present each of these agents in more detail.

5.1.1. Watcher Agents

The watcher agents collect information for the needs of the applications. For example, in an application which presents awareness information about the presence of co-workers, watcher agents can collect information about user activity and location from different sources (PC-activity, pressure sensitive floor mats, speaker recognition results), tasks users are doing at the moment, who are talking in the same room, and who has reserved calendar events.

Watcher agents use either the common Information Storage or external databases to gather the information. The information may be formed and stored by another parts of the application, or external applications. When necessary, the watcher agents can transform this information into a more conceptual form better understood by other agents. For example, they might clean excess details from recognition results, so that only the parts needed by logic agents are given to them. In the context of a wider collection of applications, which form a pervasive computing environment, it is natural to distribute information collection and presentation as separate and independent applications, which still use the common Information Storage.

5.1.2. Logic Agents

Logic agents use information gathered by watcher agents, and use it for reasoning of planning and initiating actions at a conceptual application domain level. The reasoning is not related to music or sound, but to things such as when presentations should be started, to whom the presentation should be given, how much users can be disturbed, and so on. In other words, these agents act as triggers for activity. Where watcher agents are more concerned about conveying and transforming information, the logic agents use more responsibility in combining the information, making conclusions and triggering activity based on that information.

For example, logic agents understand that pieces of information like “the time is 08:00”, “the date is May 13th 2005”, “Jaakko has been recognized at the door”, “Jaakko’s previous activity was at May 12th 2005 at 19:50” together mean that “Jaakko has arrived to work” and “a sound representing Jaakko’s arrival must be added to the sound presentation.”

5.1.3. Composition Agents

The task of the composition agents is to create a conceptual sketch of the presentation, or the composition, based on what the logic agents have decided on a conceptual level to be presented. The composition agents map the application domain information into presentation domain form. Their task is to decide which instruments and tracks are used, how they are linked together compositionally and what other melodic decisions are made. Composition agents build compositions based on the conclusions the logic agents made.

For example, composition agents combine pieces of information like “Jaakko has arrived to work” and “Esa-Pekka has been at work for 40 minutes” together into information that the soundscape is composed of tracks called “Jaakko arrives to work”, “transition from Jaakko’s arrival to Esa-Pekka’s presence”, “Esa-Pekka’s presence”, “Esa-Pekka uses text processing application”, and “transition from Esa-Pekka’s text processing to the end of presentation.”

At this level, the presentation is still in a conceptual form, and the elements of the composition are thought of in the terms of the application domain. The composition is defined in layered abstractions which hide the complexity of the whole. Transition-elements are a good example of this.

5.1.4. Instrument Agents

The task of the instrument agents is to turn the conceptual tracks, instruments and effects used by the composition agents into concrete commands and parameters for the sound engine. Instrument agents create and define things like audio event requests, filenames, precise timing and condition descriptions, effects like volumes and reverbs and other concrete elements. In some cases, the same tasks can be performed by the instrument agents and composition agents, since certain effects like volume changes can be used both at compositional level and at instrument level. The main difference between compositional and instrument level is the difference between conceptual and concrete ways to describe a presentation. For example, instrument agents understand that a track called “Jaakko arrives to work” means a file with the name “bird_13.wav” must be faded-in in five seconds, after which the same file must be repeated three times at 100% volume.

When the instrument agents have done their work, the whole presentation is described in a form which the sound engine understands, and it is sent through the Information Storage to the sound engine to be played as described in Section 5.2. The engine reacts to the requests according to its state and internal rules. Further communication between the engine and the rest of the application can mean that the presentation is adjusted to suit certain conditions, like synchronization. Next we will describe the sound engines for which the agents created the presentation description.

5.2. Sound Engines

The last step of the sound presentation process is the sound engine component, which produces the final sound data stream and sends it to the audio output interface. Sound engines are independent services, and any application can contact them and send a presentation request. There is a service for every physical resource, for example a loudspeaker in a room. Technology-wise, this means a service for each physical audio output channel of any given computer’s sound card in the system setup.

The hardware interface and low level functionality can be implemented with third party software, for example Java Audio Synthesis System (JASS) [24] or Microsoft Direct Sound [17]. At the moment we use Java Sound API. Next we outline the interface, services and functionality the sound engine offers.

The sound engine is an independent, continuously running service, which can receive requests and commands, according to which it acts. The engine is generic, which means that the commands and requests it understands are not tied to any single application, but to sound manipulation and presentation in general. For example, a common command can be a request to play a track, or fade a track to a certain volume level in a certain amount of milliseconds.

The engine can also communicate back to the applications that use it, i.e., it has a callback interface. The engine responds to requests depending on their content, for example by acknowledging the received commands, informing the application of possible moments of changing the presentation, or when it would be possible to synchronize events, such as graphical animations, outside the audio presentation. In these ways the engine can be more active in helping the applications that use it. All callback communication is based on variables with which the sound engine keeps track of things it has presented and will present. A variable can be, e.g., a single track consisting of several sounds and a fade-out effect, or the time in milliseconds from the beginning of the whole composition. This is analogous to speech synthesis control techniques, especially audio to video synthesis, such as those used in “talking head” applications.

In addition to reacting to commands from outside applications, the sound engine can react to its internal events. Replies, musical change moments, synchronization points and other similar events are seen as triggers in the composition data by the engine. On the grounds of these triggers the engine changes its behavior or replies back to the applications. Triggers are embedded as parts of the compositions, commands and requests sent to the engine. They contain conditions, fulfillment of which the engine monitors. A trigger consists of two parts: A condition of a trigger can be, for example, that a track has reached a predefined point, or that the composition is playing in D-minor. A trigger also contains a command, which can be, for example, to reply back to the application or a request to further guide the behavior of a presentation. The engine might not understand the contents of the information it sends back, since it can be an extension defined by the agents and just stored as a variable in the engine. Triggers can use commands to guide the behavior of the engine for different reasons, many of which were presented in Section 4. The conditions included in triggers are not restricted only to the variables dealing the presentation, since variables of the engine’s internal behavior can also be useful. The interactivity of applications, i.e., reacting to outside events is handled by agents in other parts of the application. This was presented in Section 5.1.

The requests, commands and composition descriptions are XML-documents. They contain the actual command-words and the needed extra parameters, such as conditions and variables to be used. The parameters may contain detailed descriptions of compositions and necessary triggers. When the sound engine replies back to the other components, it uses the same XML-notation. In Sections 4.2 and 4.3 we presented why the functionality of agents and the sound engine must be described with a scripting language so that all their functionality is not hard-coded into them. Next we will take a look at how this is achieved in practice.

5.3. A Language for Agent Control

The internal functionality of the agents is partly described with a scripting and markup language. This description is loaded from configuration files when the application is launched. The same agent can be used for different purposes with different kinds of configurations.

It is important to leave room for modifications and variability in the agents, so they can be easily altered during the design phase of the presentation without recompiling source code. The designer or composer might need a large collection of agents to describe all of the functionality he or she wishes from the application. The purpose of the scripting language is to have a common ground for composers and application designers to work together. The scripting language must be easy enough for people without much experience in programming to describe their compositions in an interactive manner, and also for the software implementers to understand the compositional elements in unambiguously defined ways. Like Korpipää [11], we think that "context should be human-understandable." We see three levels of users for the architecture: API-level architecture developers, application developers and composers, who mainly use the scripting and markup languages, and application end users, which could use a yet undefined end user development (EUD) tool, built on top of the scripting and markup languages.

The presentation descriptions are stored in the Information Storage in XML-format. The way the agents interpret these descriptions depends both on hard-coded and configured functionality. The final meaning of each part in the script is always defined both in the agents and in the sound engine. Since the scripting language is extensible, agents will handle those parts of the composition description they can understand. This also makes the agents reusable. Just like the agents, the presentation description is also constructed in an object-oriented manner.

Excluding instrument agents, the purpose is to describe things (sounds, effects, variables) in conceptual and symbolic terms that relate to the application domain (for example "Esa-Pekka arrives to work"), and not in technical terms (filenames etc.). They also hide the complexity of the technical descriptions. This is important in order to have common ground for application designers and composers. This is possible by offering a set of agents which implement the basic functionality, and which can be used for different purposes with the scripting language. Agents scripted in different manners can then be combined and chained in order to achieve more complex functionality. This does not exclude the possibility to implement new agents. It depends on the application domain what kinds of agents are required. Complicated signal processing, for example, is not necessary in all applications.

Although much functionality can be presented as simple configuration information, there is also a need for a more or less fully capable programming syntax with Boolean logic and control structures. This is a shared grey area between agent capability and script capability. There are many tried and tested markup and scripting languages which would fulfill the needs of many soundscape descriptions, either from a musical or functional viewpoint [18; 20; 25; 26].

6. DISCUSSION AND FUTURE WORK

We presented a generic model (Figure 1) and an implemented application for supporting group awareness through auditory presentations. We discussed the importance of calm and unobtrusive presentations, and ways of achieving these through continuous and consistent soundscapes.

We presented an architecture based on these requirements. Winograd [27] set criteria for comparing architecture models for context aware applications, namely efficiency, configurability, robustness, and simplicity. He argues that the blackboard architecture is the most suitable model for context management. Our architecture aims to extend this argument to also encompass presentation management of soundscape applications.

Our architecture is designed for building applications which present awareness information through soundscapes. The architecture consists of agents, an Information Storage and a sound engine. The agents are compact and numerous and they distribute the task of information gathering, reasoning, composition control and instrument control. The agents are generic, reusable and extensible because of the object-oriented approach we took in designing the architecture. The dynamic choice of agents, as described in Section 5.1, gives them flexible capabilities to function, as opposed to a pipelined structure. There are many examples of this in speech applications. Our approach gives a basic structure which can be modified if needed. The functionality of the agents is configurable through scripting. The components use a control language to communicate, which is also designed to support distribution, reuse and extensibility. Complexity is handled through hierarchical abstraction and distribution. A strong connection between the sound presentation architecture and external applications is provided by the underlying framework, and communication is easy through the common Information Storage.

We plan to continue the development of the architecture described in this paper to integrate it with more applications and information sources. When a reliable level of implementation is reached, we have planned to make the architecture available under, e.g., L-GPL license. We have also compared several scripting and markup languages for music description, and we are planning to further specify our control language to take advantage of their tried and tested best conventions. We also plan to concentrate on two-way distribution of data gathering and presentation definition; the end-users of the soundscapes must have a better control over what information they give over. This also gives them control over personal preferences in the presentation content, structure and style. These are important trust and information security issues. In addition, people have differing tastes and information needs, although only a few have the musical skills to design working soundscapes all by themselves. Fully automatic and non-customizable functionality is rarely useful [11], so we plan to expand configurability towards end-user development in the future. This would give a better opportunity for the user to get feedback from inferred contexts, adjust the information and inferred actions, maintain flexibility, and avoid risky automatic actions [28].

7. REFERENCES

[1] V. Bellotti and K. Edwards, "Intelligibility and accountability: Human considerations in context-aware systems," *Human-Computer Interaction*, Vol. 16, No. 2, 3 & 4, pp. 193 – 212, 2001.

[2] K. Mäkelä, J. Hakulinen, M. Turunen, "The Use Of Walking Sounds In Supporting Awareness," in Proc. International Conference on Auditory Display (ICAD), Boston, MA, USA, July 2003, pp. 144-147.

[3] A. Kainulainen, M. Turunen, J. Hakulinen, E.-P. Salonen, P. Prusi, L. Helin, "A Speech-based and Auditory Ubiquitous Office Environment," in Proc. 10th Int. Conf. on Speech and Computer (SPECOM), Patras, Greece, October 2005, pp. 231 – 234.

[4] K. Mäkelä, E.-P. Salonen, M. Turunen, J. Hakulinen, R. Raisamo, "Conducting a Wizard of Oz Experiment on a Ubiquitous Computing System Doorman," in Proc. International Workshop on Information Presentation and Natural Multimodal Dialogue, Verona, Italy, December 2001, pp. 115 – 119.

[5] M. Turunen, J. Hakulinen, K.-J. Räihä, E.-P. Salonen, A. Kainulainen, P. Prusi, "An architecture and applications for speech-based accessibility systems," IBM Systems Journal, vol. 44, no. 3, pp. 485-504, 2005.

[6] P. Dourish and V. Bellotti, "Awareness and Coordination in Shared Workspaces," in Proc. ACM Conference on Computer-Supported Cooperative Work (CSCW), Toronto, Canada, October-November 1992, pp. 107-114.

[7] T. Matthews, T. Rattenbury, S. Carter, A. Dey, J. Mankoff, "A Peripheral Display Toolkit," *Intel Research Berkeley, Technical Report IRB-TR-03-018*, 2003.

[8] C. Wisneski, H. Ishii, A. Dahley, M. Gorbett, S. Brave, B. Ullmer, P. Yarin, "Ambient Displays: Turning Architectural Space into an Interface between People and Digital Information," in *Proc. 1st Int. Conf. on Cooperative Buildings (CoBuild'98)*, Darmstadt, Germany, February 1998, pp. 22–32.

[9] L. Lövstrand, "Being Selectively Aware with the Khronika System," in *Proc. The 2nd European Conference on Computer Supported Collaborative Work (ECSCW'91)*, Amsterdam, The Netherlands, September 1991.

[10] M. Beaudouin-Lafon and W.W. Gaver, "ENO: synthesizing structured sound spaces," in *Proc. The 7th annual ACM symposium on User interface software and technology (UIST'94)*, Marina del Rey, CA, USA, November 1994, pp. 49-57.

[11] P. Korpipää, *Blackboard-based software framework and tool for mobile device context awareness*. Dissertation, VTT Publications 579, VTT Electronics, Espoo, Finland, 2005.

[12] D. Temperley, "An Algorithm for Harmonic Analysis," *Music Perception*, vol. 15, no.1, pp. 31-68, 1997.

[13] C.L. Krumhansl, "The Geometry of Musical Structure: A Brief Introduction and History," *ACM Computers in Entertainment*, vol. 3, no. 4, pp. 1-14, October 2005.

[14] B. Eno, "Generative Music," in *Imagination Conference*, San Francisco, June 1996. (talk) Available in http://www.inmotionmagazine.com/eno1.html [3.6.2005]

[15] M.Z. Land and P.N. McConnell, "Method and Apparatus for Dynamically Composing Music and Sound Effects Using a Computer Entertainment System," *U.S. Patent 5,315,057*, May 1994.

[16] Microsoft Corporation, "Composing Music for Interactive Titles: An Overview of Direct-Music Producer," 1998. Webpage: http://msdn.microsoft.com/library/deafult.asp?url=/library/en-us/dnmusic/htm/interact.asp [3.6.2005]

[17] Microsoft Corporation, "Microsoft Direct Sound," 2005. Webpage: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/htm/directsound.aps [3.6.2005]

[18] C. Grigg, "Preview: Interactive XMF – A Standardized Interchange File Format for Advanced Interactive Audio Content," in *115th Audio Engineering Society Convention*, New York, NY, USA, October 2003.

[19] G. Wang and P.R. Cook, "Chuck: A concurrent, on-the-fly audio programming language," in *Proc. Int. Computer Music Conf. (ICMC)*, Singapore, September-October 2003, pp. 219-226.

[20] P. Taylor, and A. Isard, "SSML: A speech synthesis markup language," *Speech Communication*, vol. 21, no. 1-2, pp. 123-133, February 1997.

[21] A. Dey, G. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction*, Vol. 16, No. 2, 3 & 4, pp. 97 – 166, 2001.

[22] L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *ACM Computing Surveys*, vol. 12, no. 2, pp. 213-253, June 1980.

[23] E.-P. Salonen, M. Turunen, J. Hakulinen, L. Helin, P. Prusi, A. Kainulainen, "Distributed Dialogue Management for Smart Terminal Devices," in *Proc. Interspeech 2005*, Lisboa, Portugal, September 2005, pp. 849-852.

[24] K. van den Doel and D. Pai, "JASS: An Audio Synthesis System for Programmers," in *Proc. International Conference on Auditory Display (ICAD)*, Espoo, Finland, July-August, 2001, pp. 105-154.

[25] B. Lucas, "VoiceXML for web-based distributed conversational applications," in *Communications of the ACM*, vol. 43, no. 9, pp. 53-57, September 2000.

[26] G. Haus and M. Longari, "Towards a Symbolic/Time-Based Music Language Based on XML," in *Proc. First Int. Conf. on Musical Applications using XML (MAX2002)*, Milan, Italy, September 2002, pp. 38-46.

[27] T. Winograd, "Architectures for context," *Human-Computer Interaction*, Vol. 16, No. 2, 3 & 4, pp. 401 - 419, 2001.

[28] S. Greenberg, "Context as a dynamic construct," *Human-Computer Interaction*, Vol. 16, No. 2, 3 & 4, pp. 257 – 268, 2001.