Distributed Implementation of a Self-Organizing Appliance Middleware

Michael Hellenschmidt[(1)]

[(1)] Fraunhofer Institute for Computer Graphics
Fraunhoferstr. 5, 64283 Darmstadt
Michael.Hellenschmidt@igd.fraunhofer.de

Abstract

A middleware for real ad-hoc cooperation of distributed device ensembles must support self-organization of its components. Self-organization means that the independence of the ensembles' components is ensured, that the ensemble is dynamically extensible by new components and that real distributed implementation is possible. Furthermore the data-flow of messages within the ensemble may not be statically determined. This article presents the distributed implementation of the *SodaPop* model for distributed device ensembles of physical heterogeneous devices as well as the distributed handling of conflict resolution strategies that guarantee the data-flow even if there are competing components. The proposed approach relies on the principle of device representatives. Here physical devices host their components and disburden them from communication and service composition strategies.

1. Introduction

Rather popular scenarios for Ambient Intelligence [1, 2] illustrate the visions of smart conference rooms or smart living rooms where devices are able to cooperate in an ad-hoc fashion. Well-established examples are the Easy Living project from Microsoft [3], the Interactive Workspaces Project [4] from Stanford University or the Intelligent Classroom [5] from Northwestern University. But those smart environments from the various research labs are usually assembled from devices and components whose functionality is known to the developers. Furthermore, in systems with distributed devices, the data flow from device to device is determined for every use case. Consequently the intelligence of Ambient Intelligence prototypes and demonstrators is carefully handcrafted.

This is not possible for dynamic scenarios, where people come together in a meeting room for example, each of the participants bringing with own personal devices, or where people are buying new devices for extending their existing entertainment device ensembles. A scenario that outlines the vision of intelligent environments that were built up ad-hoc by cooperating devices is the example of an ad-hoc meeting where People meet at a perfectly average room. All of the participants bring their own notebook computers, at least one brings a projector and the room has some light controls. So it would be possible for this spontaneous ensemble to provide the same assistance as a fixed conference room. This kind of Ambient Intelligence requires more than setting up a central control application in advance. It requires the ability of the devices to autonomously configure themselves into a coherently acting ensemble that is fully distributed.

Johanson from Stanford University also points out [4] that "users should only have to plug in a device or bring it into a physical space for it to become part of the corresponding software infrastructure. User configuration should be simple and prompted by the space. …The logical extension of this is to allow ad hoc interactive workspaces to form wherever a group of devices are gathered."

Obviously software infrastructures are needed that are distributed implemented and that allow a true self-organization of the device ensembles that are connected in an ad-hoc fashion. In order to take a step ahead to this vision the project DynAMITE [6] develops a decentralized middleware for self-organizing device ensembles on basis of the middleware model *SodaPop* [7, 8]. This article specifies the distributed implementation of this middleware concept. Therefore the next chapter reviews the requirements for self-organizing ensembles that come up while looking at the underlying scenarios. In Section 3 the core concepts of *SodaPop* are illustrated (for a detailed specification please refer to [7, 8]). *SodaPop* introduces a solution proposal for a software infrastructure that supports such heterogeneous ad-hoc device ensembles. Section 4 then outlines the distributed implementation of our approach and explains conflict resolution mechanisms among distributed devices with the principle of device representatives. After some explanations about the underlying communication infrastructure and the reflection of the related work this article ends with a discussion and an outline of our next steps.

2. Requirements

The challenge of self-organization as indicated in the introduction of this article distinguishes two different aspects:

- Architectonic Integration: this refers to the integration of a (new) device into the communication patters of an existing device ensemble. This refers also to the ad-hoc assembly of a device ensemble from heterogeneous standalone devices.
- Operational Integration: this describes the aspect of making new functionalities that are provided by a (new) device available to the user.

Obviously operational integration means a form of service discovery transparent to all devices. It can be realized based on an explicit modeling of the semantics of device operations as precondition / effect rules that have to be defined over suitable environment ontology (see [9] for a detailed reflection on this topic). One has to bear in mind that operational integration means more than to make a graphical user interface available for the user like the approaches in Jini [10] or HAVi [11].

This article concentrates on the aspect of architectonic integration. While looking at typical scenarios where devices have to cooperate in an ad-hoc fashion one identifies (see [7, 8] for more details) that each device should also be able to act stand-alone and has to be independent. Additionally distributed implementation has to be supported to guarantee that there may not be any kind of central component (because a central controller is a contradiction itself to the demand of ad-hoc self-organization) and the devices should be

exchangeable. Furthermore, transparent service arbitration should be provided. Ambient Intelligence can be identified (see [7, 8] for more details):

Only if all requirements are met intuitive scenarios like the ”Plug and Play” of new devices into an existing device ensemble, and the build-up of a device ensemble in an ad-hoc fashion (with no discussion where a central router should be started) is possible. The requirement that devices should be able to work stand-alone corresponds to the user's experiences and expectations of the every day usage of conventional devices.

3. Principles of the *SodaPop* model

This section should outline the core ideas of the *SodaPop* model (for details we refer to [7, 8]). Each device that is able to interact with users (like TV sets by buttons or remote controls) and that is able to change the user's environment (by rendering a medium for instance) possesses a kind of event processing. Figure 1 outlines possible processing stages and a specific event processing pipeline. Usually devices have a user interface that translates physical user interactions to events. An Interpreter component then is responsible for determining the appropriate goals, which are translated into function calls by a Control Application. The Actuators then are physically executing this function calls [12].

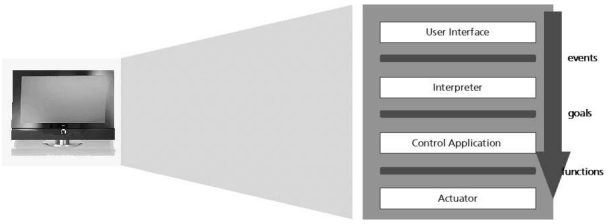

Figure 1: The internal data-flow of a standard device.

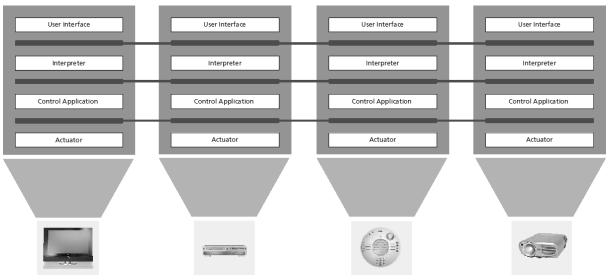

Figure 2: The devices share the interfaces after they are extended across them.

If some devices are plugged together (see figure 2) the interface between the individual processing stages can be extended across multiple devices. That means, after turning the private interfaces between the processing stages in a device into public channels, Interpreter components from one device are able to ”see” events from other devices. Or the Control Application of one device is able to interpret goals that are made by other devices.

Obviously if all components are able to “see” the messages of the other components that are subscribed to a channel, some conflicts will come up which component(s) are allowed to process the message. Those conflicts of competing components have to be solved by conflict resolution strategies, which are part of the channels message handling capabilities. The procedure of conflict handling within a channel is illustrated by figure 3. In a nutshell, *SodaPop* differs between two types of components:

Channels read single messages and map them to (multiple) messages. Therefore conflict resolution strategies are used that are evaluating the channel subscribers' utility value functions, decomposing the messages and delegating them to the receiver components (see figure 3). How a channel determines the effective message decomposition and how it chooses the set of receiving consumers is defined by the individual channel's decomposition strategy (that is eventually based on the channel's ontology).

Transducers represent the components in figure 1 and figure 2. Transducers are able to read one or more messages and are able to map them into appropriate output messages (e.g. events are mapped into goals). When subscribing to a channel, a transducer declares: the set of messages it is able to process and how well it is suited for processing certain messages. For this reason the transducer makes its utility value function available to the channel(s) it is connected to.

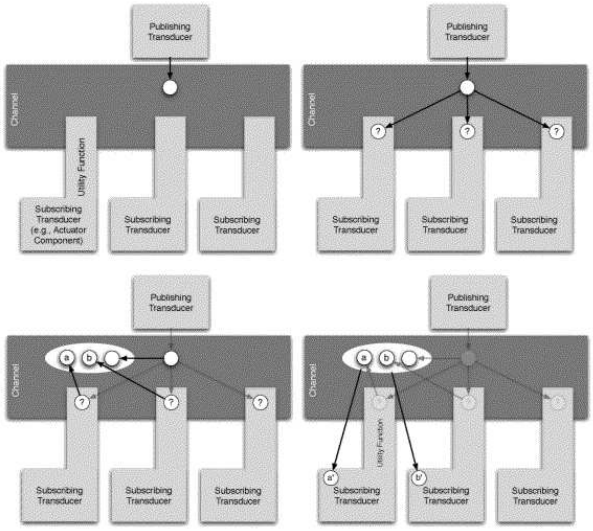

Figure 3: The basic mechanism of conflict resolution strategies evaluates the consumer utility values, decomposes the message and delegates it to the receiver components.

After a common set of channels as well as appropriate conflict resolution mechanisms are identified an architectonic integration of devices and components could be achieved by means of the *SodaPop* principles. In [12] a Generic Topology for Ambient Intelligence is identified. It consists of four levels of components (Interaction, Interpretation, Strategy Assistants, and Actors, see also figure 1 and figure 2). Also some possible conflict resolution strategies within the domain of home entertainment and the domain of lecture rooms are described A conflict resolution strategy that decomposes single messages with amodal information into multiple messages with modal information (e.g. system output information that is split up into graphical and voice output in dependence of the abilities of the connected consumer components) is explained in detail in [13].

4. Distributed Implementation

In order to make the distributed implementation of the self-organizing middleware model *SodaPop* possible, it might again be helpful to look inside the physical devices that should be supported. For devices like the one that is illustrated in figure 1 the implementation seems to be trivial. The User Interface sends its events directly to the Interpreter. After that the Interpreter forwards its goals to the following Control Application. And finally the Control Applications sends the functions calls to the Actuator.

But what will happen if a vendor wants to sell two standalone devices in one physical unit? Or in other words: How can a physical device be internally managed if it consists of two logical devices?

Now the Interpreter components of both logical devices (figure 4 illustrates an example of a combined TV set-DVD device) see all events that come from the different User Interfaces. Of course it would be reasonable if only one Interpreter component infers the user's goals. And of course if later on only one Control Application schedules the appropriate functions. In order to provide this the channel has to apply the necessary conflict resolution strategies.

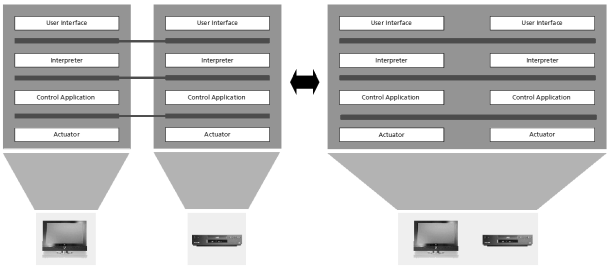

Figure 4: Two stand-alone devices (left) are combined to one physical device that then consists of two logical devices (right).

Obviously the channel is created by its connected transducers and thus the participating transducers have to carry out the conflict resolution strategy among them. To provide this functionality, some principal questions have to be answered:

- Is it possible to apply conflict resolution strategies cooperatively (among the participating transducers)?
- Or is there a way to choose one transducer that should apply the conflict resolution strategy alone?

And furthermore, if it is possible to find solutions at least for one of these questions:

- If the execution of conflict resolution strategies is possible in a cooperative way - Will it also be possible to apply the found approach across real distributed physical devices? That means: Is it possible to find applicable methods for parallel processing not only among distributed applications but also among distributed processors?
- If one transducer can be chosen to apply exclusively the appropriate conflict resolution strategy, will it also be able to choose one transducer among distributed physical devices?

The scenarios (section 1) and the resulting requirements (section 2) demand the independence of each physical device, not of each component that runs on these devices. That means it is reasonable to increase the granularity from logical components (e.g. a User Interface or an Interpreter component) to real physical devices (e.g. a TV set that is the host for its different components). This is obviously according to the user's expectations. The user wants to combine physical devices and not logical components.

Because the physical device is the smallest entity within a device ensemble it can run one instance of a so-called *SodaPop*-Demon. Consequently the *SodaPop*-Demon hosts all different transducers of its physical entity (see figure 5).

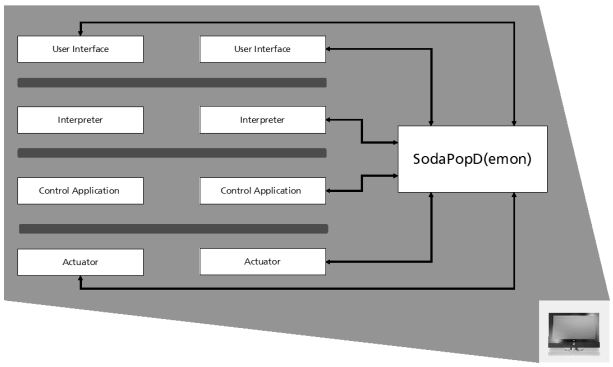

Figure 5: Each physical device runs one SodaPop-Demon instance where all logical components, the transducers, are connected to. The SodaPop-Demon is also the host for the channels that are defined by the multiple transducers.

Once a transducer runs:

- It connects to the device's own *SodaPop*-Demon by declaring the descriptions of the channels it wants to participate
- It indicates for each channel, whether it wants to listen to channel messages or it wants to write to the channel
- And it declares the set of messages, it is able to process

If now a User Interface component wants to send an event to the channel it is connected to, it will send a message to its *SodaPop*-Demon. The message contains the event itself together with some information about the receiver channel and the sender itself. After that the *SodaPop*-Demon contacts all transducers that are subscribed as listeners to the corresponding channel to evaluate their utility value function according to the initial message. After the *SodaPop*-Demon had collected all utility value function results it starts to execute the channel's conflict resolution strategy. Finally the *SodaPop*-Demon delegates the decomposed message(s) to the receiver transducer(s). In order to avoid traffic between the transducers and the *SodaPop*-Demon we differ between static and non-static utility value functions. In case a utility value function is static, its values are handed over to the *SodaPop*-Demon when the transducer starts and connects.

Thus, the *SodaPop*-Demon can use its own look-up table instead of causing message traffic. In general utility value functions are non-static. They are dependent on the current state the transducer belongs to when its utility value function is evaluated. An example is a rendering component for media that already renders a movie. Of course at this moment it will raise lower utility values than another rendering component whose resources are all available. Some consequences of this approach should be mentioned:

- The channels now turned into virtual entities. Consequently a channel descriptor defines the name of a

logical group to which transducers correspond to according to the ontology that is semantically used for the communication. Also the channel descriptor defines the effective conflict resolution strategy that has to be used in case of competing components.

- Messages between transducers and channels are sent via a *SodaPop*-Demon. A *SodaPop*-Demon is the container for the device's channels as well as the container for its different components and their utility value functions.
- All strategies that have to be executed to guarantee the information flow inside a physical device are applied by the *SodaPop*-Demon of the device.

The consequence that a *SodaPop*-Demon is obviously a central component for each physical device does not limit the scenarios and the requirements, because we want to achieve plug and play of devices and that device ensembles are able to interoperate in an ad-hoc fashion. The fact that each entity of a device ensemble (that means each device) runs its own service isn't any limitation at all.

4.1. The Principle of Device Representatives

After the introduction of the principles and the functions of the *SodaPop*-Demons that correspond to single devices this section explains how the *SodaPop*-Demons can be used as representatives of their device, their channels and their logical components in heterogeneous device ensembles.

Figure 6 illustrates the principles of the distributed implementation. The *SodaPop*-Demons as the representatives of their devices' channels and transducers build up groups where peer-to-peer communication is possible (see 4.3). Each of the group represents a certain channel. Consequently a *SodaPop*-Demon enters a group when an own transducer connects to the corresponding channel, and a *SodaPop*-Demon leaves a channel, when the last own transducer disconnects from the corresponding channel. If a group that corresponds to a channel does not exist the responsible *SodaPop*-Demon will open up an appropriate group.

Amongst a group the direct addressing from *SodaPop*-Demon to *SodaPop*-Demon is possible (unicast) as well as multicasts from one *SodaPop*-Demon to all members of a group. Therefore only one restriction exists: Each *SodaPop*-Demon owns a one-to-one identification number to guarantee reliable point-to-point communication (this can be done by using individual manufacturer numbers or rather MAC numbers).

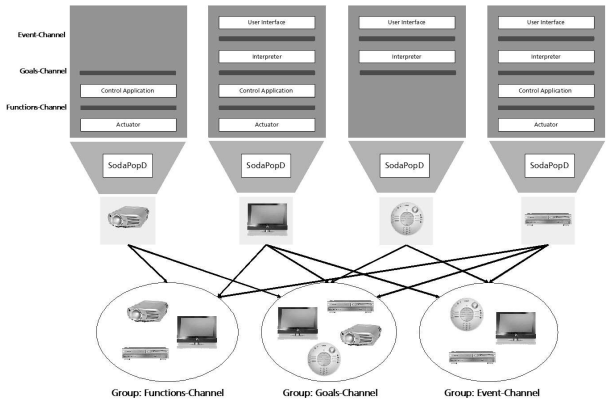

Figure 6: The SodaPop-Demons of the different devices build up three different groups that correspond to the three defined channels. The different SodaPop-Demons are symbolised by their device icons.

4.2. Decentralized Conflict Handling

Now the delivering of messages from component to component reduces to the challenge to find an appropriate *SodaPop*-Demon that hosts a qualified transducer. Conflict resolution mechanisms must be applied by the *SodaPop*-Demons that build up the group that corresponds to the channel where the message is received. And furthermore the *SodaPop*-Demons as representatives of their transducers take part in the competition for messages at the same time. But decentralized conflict handling without any central or salient group member needs some conventions:

- Every group member needs the same information: *SodaPop*-Demons announce the number of listener- and writer-transducers they represent in the group as well as their individual identification number
- Each *SodaPop*-Demon that hosts a listener transducer (that means a transducer that wants to consume messages) must be able to execute the conflict resolution strategy that corresponds to the represented channel. The reason for this restriction is intuitively understandable: If the calculating capacity of a *SodaPop*-Demon is sufficient to host a transducer that can interpret messages and infer user goals for instance, it will have also the capacity to run substantial strategies. Consequently simple sensor devices like RFID marker or motion detectors must not provide too much calculating capacity because they are only the sources of events and not the consumers.

The communication mechanisms inside a group of *SodaPop*-Demons takes place in the following way:

1. If a transducer sends a message to a channel (e.g. the User Interface of the remote control in figure 6) the corresponding *SodaPop*-Demon broadcasts this message to the other members of this (channel) group.
2. After receiving the message each *SodaPop*-Demon will evaluate the utility value functions of its (listener-) transducers that are connected to the corresponding channel and will collect all utility values.
3. Then each *SodaPop*-Demon broadcasts the following information to its group participants:
    - the collection of all utility values of its transducers
    - its own identification number
    - a number *n* between 0 and 1 to declare how well it is suited to provide the conflict resolution mechanism (Note: if the *SodaPop*-Demon owns listener transducers it has to provide the necessary "intelligence" to execute the corresponding conflict resolution strategy).
    - and a time *T* that indicates the length in time when the result of the conflict resolution strategy at the latest will be broadcasted to the other group members.
4. Each *SodaPop*-Demon receives the broadcasted information of the other *SodaPop*-Demons of its group and thus all group members own all utility values of all connected (listener-) transducers.
5. The *SodaPop*-Demon that offered the highest number *n* starts to execute the channel's conflict resolution strategy and broadcasts the results to the other group member. The results are the decomposed message and the identification number(s) of the receiver transducer(s).
6. Each *SodaPop*-Demon receives the broadcasted results and - in case it hosts one or more of the receiver

transducers - forwards the decomposed message(s) to its transducers.

The process will restart at point 3, if all *SodaPop*-Demons offer the number 0 for *n*; if two or more *SodaPop*-Demons offer the same number *n* - and *n* is the highest number that is offered; or if the time *T* is elapsed without any results of the *SodaPop*-Demon that should execute the conflict resolution strategy (because that could indicate that the corresponding *SodaPop*-Demon has left the group or something other unseen had happened). The task to choose one *SodaPop*-Demon to execute the conflict resolution strategy is well known as the Leader Election Problem [14].

4.3. Underlying Communication Infrastructure

The underlying communication infrastructure for the described decentralized middleware has to fulfill the following requirements:

- Ensure peer-to-peer communication without any central components (except for applications that could run autonomously on each physical device)
- Ensure the dynamical build-up of groups
- Ensure unicast and broadcast amongst group members.

In DynAMITE we chose to apply the JXTA-technology [15] as well as the UPnP-(Universal Plug and Play)-technology [16]. Both can provide the necessary peer-to-peer communication mechanisms whereas JXTA supplies the software engineer with comfortable Java (and C) application programming interfaces. In contrast UPnP has a fast increasing community and it is expected that UPnP services will be provided in many devices in the future.

5. Related Work

A middleware for the visions of Ambient Intelligence must provide complete decentralized communication among its components. Furthermore to provide extensibility and exchangeability the middleware must be able to execute conflict resolution strategies to guarantee reasonable data-flow even if there are competing components. Different technologies and approaches face single aspects of the mentioned requirements. Jini [10], HAVi [11], JXTA [15] and UPnP [16] make the communication between devices from different vendors possible. Unfortunately no conflict resolution mechanisms - apart from graphical user interfaces - are provided. Some agent technologies are well known like SRI's Open Agent Architecture (OAA) [17], the Galaxy Communicator Architecture [18] or INCA [19]. Galaxy uses a centralized hub-component that owns routing rules that determine the data-flow whereas the OAA uses prolog-based mechanisms that are located in special meta-agents. That is alike the Jaspis framework [20] that uses evaluation agents for the evaluation of the quality of possible addressee agents. INCA uses a central component for registering components and for delivering messages. The abstract system architecture of AMIGO [21] recognizes the needs for service discovery and service composition strategies – as offered by the channels that are transparent for each developer – but each component is responsible for the application of such strategies. Thus the component developer is responsible for both: the implementation of a component that provides certain functionalities as well as appropriate service composition mechanisms. Consequently the reliability and transparency will decrease. The world of agent communication seems to be split in two halves. On the one side the peer-to-peer communication world, where all components broadcast messages or communicate directly by using fixed addresses. And on the other side the world with central components where hand-crafted routing rules are applied to the communication process. In both worlds the dynamic extensibility and self-organization of device ensembles seems to be difficult.

6. Current State and Next Steps

While [7, 8, 12] explain the principles of the *SodaPop*-model for the realization of self-organizing device ensembles and identify certain communication patterns and define some conflict resolution strategies that make the built-up of device ensembles possible, this paper specifies the fully distributed implementation of the *SodaPop*-model. The presented distributed implementation abandons any central component and bears down the disadvantages of the peer-to-peer world as well as of the world with central (routing) components. With the approach to define representatives for the multiplicity of components of each physical device as one peer and the application of conflict resolution mechanisms among each communication group (we name it channel) we reached the required self-organizational abilities. We are using a strict definition of devices, of components and of channels and thus are able to define on which level of granularity which communication strategies are needed. First applications and demonstrators are available. Elting [22] presents the built-up of a living room on top of *SodaPop* that consists of a TV, a digital picture frame and a PDA. Here the dynamic coordination and cooperation of different output devices is demonstrated. Currently the project is defining the semantic of the messages and the utility values [23]. Here, as a basis, an extended syntax of Universal Plug and Play [15] will be used. Also the project web page offers some demonstrators for download [6].

Nevertheless the dynamic in data flow needs more communication traffic than in other solutions. Obviously the evaluation of the transducers' non-static utility value functions or the broadcasting of information among the channel group members needs more communication traffic than simple peer-to-peer approaches. Example: Two devices (a and b) that host two different components each (a hosts the components a1 and a2 and b the components b1 and b2). If component b2 is the final addressee of a1's message there will be at least 8 communication messages between the different *SodaPop*-Demons and between the *SodaPop*-Demons and their components (see 4.2 and the details about the communication mechanisms). A hard-wired peer-to-peer system will need only one communication message. We think that this is absolutely tolerable in order to achieve a fully distributed implementation of our middleware approach – and the dynamic benefits of it.

In this article we concentrate on the presentation of the decision process of one message to one or more receiver transducers (see section 4.2) within a *SodaPop*-Demon group. This corresponds to a *1:1* respectively a *1:n* mapping of messages. That means one message is forwarded to one or more receiver transducers. Our next steps are the implementation of a *n:1* and a *n:m* mapping (consequently that means, that a sequence of n messages is forwarded to the same transducer respectively to m transducers). That is often the case if sequences of user interactions should result in only one user goal. Also some experiments concerning the amount of supported devices are still pending. But in our opinion the traffic of messages and also the need to execute complex

conflict resolution mechanisms in every-day scenarios meet reasonable real time requirements.

Acknowledgements

The work presented in this paper was funded by the German ministry for education and research (BMB+F) under the grant 01 IS C27 A of the project DynAMITE.

References

[1] Aarts, E. (2004) *Ambient Intelligence: A Multimedia Perspective*, in: *IEEE Multimedia*, p. 12-19.

[2] Ducatel K., Bogdanowicz M., Scapolo F., Leijten, J., and Burgelman J.-C. (2001) *Scenarios for Ambient Intelligence 2010, ISTAG Report, European Commission*, Institute for Prospective Technological Studies, Seville, available from: ftp://ftp.cordis.lu/pub/ist/docs/istagscenarios2010.pdf

[3] Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer S. (2000) *Easy Living: Technologies for Intelligent Environments*, in: *Handheld and Ubiquitous Computing* (Sep. 2000)

[4] Johanson, B., Fox, A., and Winograd, T. (2002) The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms, in: IEEE Pervasive Computing Magazine 1(2), April-June 2002

[5] Flachsbart, J., Franklin, D., and Hammond, K. (2000) *Improving Human-Computer Interaction in a Classroom Environment using Computer Vision*, in: *Proceedings of the Conference on Intelligent User Interfaces.*

[6] DynAMITE - Dynamic Adaptive Multimodal IT Ensembles, available from: http://www.dynamite-project.org

[7] Heider, T., and Kirste, T. (2002), *Architecture consideration for interoperable multi-modal assistant systems*, in: *Proc. 9th Intern. Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS 2002)*, Rostock, Germany

[8] Hellenschmidt, M., and Kirste T. (2004) *SodaPop: A Software Infrastructure Supporting Self-Organization in Intelligent Environments*, in: *Proc. of the 2nd IEEE Conference on Industrial Informatics, INDIN 04*, Berlin, Germany, 24 - 26. June, 2004.

[9] Heider, T., and Kirste, T. (2002) *Supporting goal-based interaction with dynamic intelligent environments*, in: *Proc. 15th European Conference on Artificial Intelligence (ECAI 2002)*, Lyon, France

[10] Jini, Sun Microsystems, available from: http://wwws.sun.com/software/jini/, 2003

[11] HAVi, Inc., The HAVi Specification - Specification of the Home Audio / Video Interoperability (HAVi) Architecture - Version 1.1, http://www.havi.org, 2001

[12] Hellenschmidt, M., and Kirste, T. (2004) *A Generic Topology for Ambient Intelligence*, in: *Proc. of the Second European Symposium on Ambient Intelligence (EUSAI 2004),* Eindhoven, the Nederlands, November 8 - 10, 2004

[13] Elting, Ch., and Hellenschmidt, M. (2004) *Strategies for Self-Organization and Multimodal Output Coordination in Distributed Device Environments*, in: *Baus, Joerg (Ed.) et al.: Proc. of the Workshop on Artificial Intelligence in Mobile Systems 2004 (AIMS)*, Saarbruecken, p. 20-27

[14] Feige U. (1999) Noncryptographic selection protocols, in: *Proceedings of 40th FOC*S, p. 142-152, 1999

[15] The JXTA Project, Sun Microsystems, available from: http://www.jxta.org, 2003.

[16] The Universal Plug and Play Forum, Contributing Members of the UPnP(TM) Forum, available from: http://www.upnp.org, Mar 2005.

[17] Martin, D.L., Cheyer, A.L., and Moran, D.B. (1999) *The Open Agent Architecture: A Framework for Building Distributed Software Systems*, in: *Applied Artificial Intelligence*, Vol. 13, No. 1-2, pp. 91-128, Jan-Mar 1999.

[18] Seneff, S., Lau, R., and Polifroni, J. (1999) *Organization, Communication, and Control in the Galaxy-II Conversational System,* in: *Proc. of Eurospeech'99*, pp. 1271–1274

[19] Truong, K.N., and Abowd G.D. (2004) *INCA: A Software Infrastructure to Facilitate the Construction and Evolution of Ubiquitous Capture and Access Applications*, in: *Proc. of the 2nd Intern. Conf. on Pervasive Computing (Pervasive 2004)*, Linz/Vienna, Austria, 2004, pp.140-157

[20] Turunen, M., Hakulinen, J., Räihä, K.-J., Salonen E.-P., Kainulainen, A., and Prusi, P. (2005) *An architecture and applications for speech-based accessibility systems*, in: *IBM Systems Journal*, Vol. 44, No. 3, 2005

[21] *AMIGO: Specification of the abstract system architecture* (2005) Deliverable D2.1, available from: http://www.amigo-project.org, April 2005

[22] Elting, Ch. (2005) *Orchestrating Output Devices – Planning Multimedia Presentations for Home Entertainment with Ambient Intelligence*, in: *Proc. of the Conference for Smart Objects and Ambient Intelligence (sOc-EUSAI 2005)*, Grenoble, France, October 12-14, 2005

[23] Kutter, O., Neumann. J., and Schmitz, T. (2005) *Extending Universal Plug and Play to support self-organizing device ensemble*, presented on the: *Workshop on Software Architectures for Self-Organization*, in conjunction with Pervasive 2005, Munich, Germany, May 11, 2005