WWICE – An architecture for In-Home Digital Networks

Heribert Baldus[a], Markus Baumeister[a], Huib Eggenhuisen[b], András Montvay[a], Wim Stut[b]
[a]Philips Research Laboratories Aachen
[b]Philips Research Laboratories Eindhoven

ABSTRACT

The transition to digital information and networking of new digital devices lead to considerable changes in the consumer electronic industry. New applications will arise, offering more entertainment, comfort and flexibility. To achieve this, complex problems in communication and distributed systems need to be solved. High requirements on stability, usability, quality and price call for new solutions.
This paper describes the concept of In-Home Digital Networks and will then in detail address the WWICE system. This new architecture provides a coherent system environment for the home. It focuses on services and applications, which can easily be accessed and controlled by the user. Application framework and middleware services of the layered software architecture efficiently support development of IHDN applications as well as flexible application control at runtime.

Keywords: In-Home Digital Network, Distributed Multimedia, Application Framework, Middleware, Graph Mapper

1. INTRODUCTION

Communication networks and distributed systems are enabling technologies for a transition towards digitalization in the home. Consumer electronic devices like TV, VCR, DVD etc. are currently in a migration phase from analog to digital technology. Digital network connections, being introduced now, allow for high-quality interconnection of consumer electronic systems, and will quickly lead to the deployment of complete *In-Home Digital Networks (IHDN)*. The future "digital home" will thus be a network of interworking devices and components; applications are realized by proper co-operation of various entities. According to the required bandwidth, IHDNs can be classified into three categories:

- Control: some kbit/sec for control information, e.g. heating, lighting
- Speech/data: some Mbit/sec for transferring e.g. speech, e-mail, www-pages
- Audio-/video entertainment: 100 Mbit/sec and more for transmission of music or video with highest quality.

In this document, we will focus on IHDNs for audio-/video applications. Thus, low bandwidth networks like Powerline or Bluetooth are not considered. Most new system approaches are based on IEEE1394[3], a low-cost, high-performance network platform for interconnecting new digital consumer devices, computers, and peripherals. IEEE1394 supports bandwidth allocation, isochronous multimedia streams, and plug&play functionality. Currently, up to 63 nodes can be connected with up to 400 Mbit/sec; higher rates and wireless connections are under development[4].
The IHDN for audio-/video applications is a distributed multimedia system, based on heterogeneous devices and system technologies. It has to support wireless and wired network technologies, devices with different functionality and performance, and an open software interface to dynamically introduce new functionality in the overall system.
This paper will give a short overview on current IHDN approaches. We will then present the WWICE system - **Window to the World of Information, Communication and Entertainment** - that provides a complete and coherent system environment for the home. After addressing requirements, we will introduce the Activity model that represents all types of applications in the IHDN. We then present the software architecture that was developed, realizing an efficient distributed multimedia system. Special focus will be on the middleware, and one of its most relevant services, the Graph Mapper. Section 5 concludes the paper and gives a short outlook.

2. RELATED WORK

The processing and communication of media streams requires specific system services, which are usually not provided by generic operating systems. The gap between the functionality offered by operating systems and the specific needs of distributed multimedia applications has to be closed by middleware components. Approaches working towards such middleware differ in the architectural layers that are covered, the services that are provided, the extensibility and the networking capabilities. Some of the most prominent approaches will now briefly be presented, as well as a couple of architectures that are comparable to WWICE.

2.1. HAVi

The Home Audio/Video Interoperability (HAVi) Architecture is being developed by eight major Consumer Electronic (CE) vendors. It aims at connecting and coordinating different digital devices. The following brief description is based on version 1.0 beta [1].

HAVi follows a two-pronged approach to provide new features to the users. First, it standardizes interconnection hardware and software and therefore preserves compatibility of CE devices from different vendors. Second, it contains mechanisms for running vendor-supplied and third-party software providing any kind of application (e.g. on-line image manipulation, web access or automatic news recording).
To enable this, HAVi distinguishes two kinds of environments: Controllers* running the HAVi stack and applications, and controlled devices, only storing the software needed to drive them. The HAVi stack (cf. figure 1) on the controllers contains

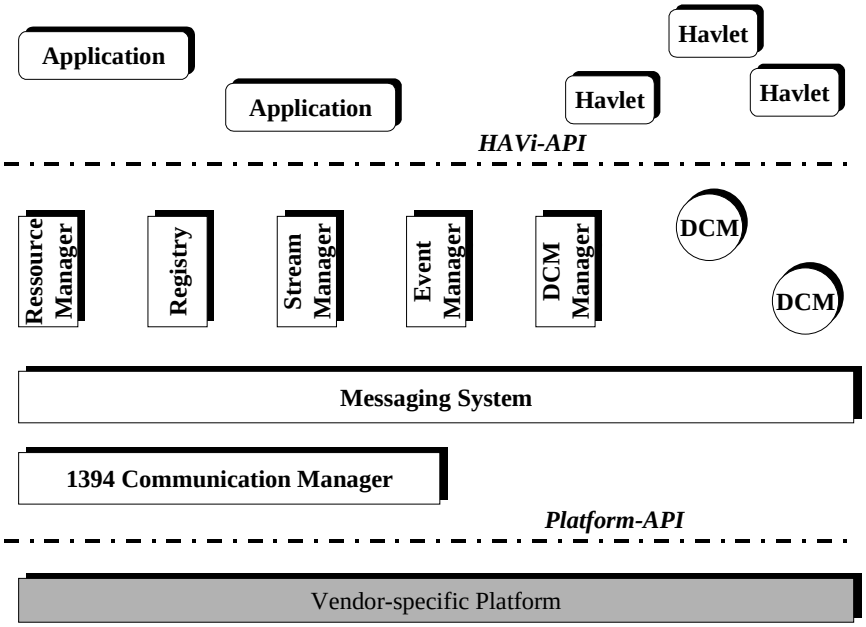

Figure 1: The HAVi stack

a communication layer abstraction (with IEEE 1394 as standard communication platform) and the middleware. The latter consists of a *Registry* to find devices and services, a *Resource Manager* to allocate and share them, a *Stream Manager* to establish multimedia streams between devices, and an *Event Manager* for distributing events to each interested party. Furthermore the Resource Manager provides the ability to schedule future actions and another component, the Device Control Module (*DCM*) *Manager*, is responsible for installing the standardized driver-like module (the DCM) for every device in the network. On top of these services reside the applications which provide user functionality. Applications and DCMs can be embedded into a controller or obtained as Java bytecode from a controlled devices or other sources (CD, Internet, etc.). Applications can display their information via two standardized GUI-Toolkits, either active with Havlets uploaded onto a controller or passive by providing rendering information to a standardized interpreter.
With its middleware layer and the standardized device drivers HAVi achieves a basic interoperability between devices of different vendors and allows applications to be written for all of them. Applications are geared to display advanced interfaces giving access to proprietary features of devices. This implies device orientation in the visions depicted in the standard [1. (chap. 1.1)] and in the functionality provided by the middleware. Applications are mainly provided by devices and device allocations always have to reference specific devices. Also it does not support the atomic establishment of multiple streams. For more sophisticated multimedia applications such features would be desirable.

2.2. VESA Architecture

The VESA Home Network Committee, a joint initiative of computer, telecom and consumer industry, is developing an overall architecture for a digital network common to all intelligent devices in the home. This comprises network, device, control and network management models with interoperating standards based on existing specifications where possible. This way, the VESA home network aims at complete interoperability of all networked devices in the home, independently of the technology of their underlying component network.
The VESA architecture[6.] supports existing network technologies by defining a backbone network and dedicated interfaces to the backbone. IEEE 1394b[4.] is used as backbone technology and IP for internetworking. Component networks connect

* Please note that in this document we will call HAVi FAVs and IAVs "controllers" and HAVi BAVs "controlled devices". HAVi LAVs, though important, are ignored to ease understanding.

devices of one and the same network technology, e.g. IEEE 1394, Powerline or RF wireless LAN. Different component networks are connected via the backbone and the corresponding interfaces. Access to external networks can be established via backbone interfaces as well as via dedicated component networks.
A common network layer unites the complete home network, comprising various physical technologies. IP protocols are used to enable control and asynchronous communication among all devices in the home. On the network layer, a device discovery service allows for dynamic integration of new devices. If available, DHCP (Dynamic Host Configuration Protocol) is used. A control browser enables the user to dynamically interact with web pages, representing the available devices. Thus, devices become visible and usable when attached. Devices can also be accessed from the external network for control purposes. For video streaming, the IEEE 1394 (and IEC 61883) protocol is used. Devices using high bit rate MPEG audio or video streaming access directly to the IEEE 1394 network, so that transmission can be performed directly on IEEE 1394 isochronous protocols.
VESA focuses on integration and cooperation of devices based on existing network technology as well as IP and IP-based higher layer protocols. This brings the benefit of more open migration paths, but will also cause considerable effort for device co-operation and dealing with various topology situations. In contrast to HAVi, standardized functional interfaces of devices ('tuner', etc.) are not defined.

2.3. Jini

Sun Microsystems' Jini™ technology is centered on the idea of locating resources or services that are somewhere in a network. It does not specifically address home networks, but it involves some scenarios that are related to a home environment. Jini is based entirely on the Java platform and relies on the existence of a network with "reasonable speed and latency".
In a Jini system, groups of users and the resources required by those users are *federated* into an image of a single distributed system. The most important concept of Jini is that of a *service*. A service is defined to be an entity that can be used by a person, a program, or another service. Typical examples for services are printing a document or translating from one data format to another. Several services can be collected together for the performance of a specific task. In order to realize locating services, a lookup service is used, which is based on the Discovery, Join and Lookup protocols.
Jini technology enables any device that can run a Java Virtual Machine to interoperate with others by offering and using services. There is however no support on an application level at all. This means that no coherent user interface is achieved in a Jini system, which would be desirable in an IHDN. Also Jini specifies only a few concrete services that can be offered in a system, so there are is no list of standard services that are provided. This implies that some a priori knowledge about the other participants in a system is needed in order to make sensible queries to the lookup service. Thus no real transparency is achieved, neither for the application programmer nor for the user.

2.4. DVB-MHP

DVB-MHP[5], the Digital Video Broadcast - Multimedia Home Platform aims at enriching the functionality of the basic digital TV platform to allow for receiving and processing interactive content. This is achieved by standardization of the specifications of the content formats, as well as the application programmers interface (API) of the applications / services to run on this platform.
DVB-MHP thus deals with processing of delivered content on specialized devices, but does not address in-home networking issues. In an IHDN, future DVB-MHP devices could be integrated like any other networked devices. The IHDN will then transport the content, the enriched AV-streams, to the corresponding appliance.

2.5. PREMO

The ISO/IEC standard 14478 - Presentation environment for multimedia objects (PREMO) defines a middleware layer to facilitiate the development of distributed multimedia applications[9]. Based on active objects and a remote invocation mechanism it defines a framework of extensible services with a focus on synchronization and device descriptions. The framework provides event-based synchronization, graph-like description of multimedia networks with virtual devices, stream and ports, as well as several subclasses of these concepts which provide standard attributes and method interfaces for device interoperability. The network description exists in two levels of abstraction: One ignores stream types whereas the other provides these types also allowing devices which split streams and provide inherent synchronization.
Though PREMO provides abstractions for interaction devices (e.g. the mouse) it is not especially concerned with user concepts. It is therefore not surprising that it follows a purely device-based approach with no provisions for hiding complexity from the user. Also, despite its rich graph structure, PREMO seems not to have automatic facilities for graph construction and manipulation.

2.6. Da CaPo++

Da CaPo++ is a middleware that aims to combine the precise specification of quality of service (QoS) requirements with security features and the use of multiple transport protocols[7]. The architecture of Da CaPo++ is based on communication modules handling internal protocol functions like checksumming or flow-control, application support modules enabling applications to access any type of communication, and transport modules supplying transparent access to different protocols. Quality of service is understood in a very broad sense, even including security features as one aspect of QoS. This enables a homogeneous communication interface for all kinds of multimedia applications.

The main concept underlying any invocation of the middleware is that of a peer-to-peer communication session, consisting of asymmetric data flows with a forward data path and a control-only return path. There is no support for more complex communication structures, leaving the task of generating such structures to the applications. Da CaPo++ also lacks device management in a way suitable for multimedia applications, as resource management is restricted to classical computer resources like memory or CPU power.

3. UNDERLYING CONCEPTS OF WWICE

The WWICE system provides an architecture for distributed digital audio/video systems supporting new multimedia applications and their user-friendliness. Based on various use cases for an IHDN, we derived common concepts advantageous to user interaction. In this section, we will describe these underlying concepts whereas the next one will cover the architecture which provides services necessary to realize these concepts.

IHDN-systems should be used by everyday people according to or in extension of their normal way of living. This implies that e.g. people don't want to be concerned about where video/audio content is actually stored, which devices are used, or in which room they currently are. The three major principles that can be derived from this observation are:

- **Network and device transparency:** The user is not aware of any details of network topology, device allocation and availability or necessary format conversions. As long as suitable devices and the required bandwidth suitable for a task are available somewhere they will be found and allocated by the system.
- **Location independence:** The user can move around the house and take with him anything he is currently using the system for (as long as the technical preconditions are fulfilled).
- **Uniform way of handling:** The user is presented a coherent user interface on all environments. While different capabilities of these environments might imply a different look, the UI concepts have to remain the same to flatten the learning curve for the user.

Since WWICE is an overall system the last requirement is easily fulfilled by defining common GUI-parts for starting, observing and managing applications and making them available on all devices. For the first two requirements, on the other hand, we introduce a new concept (cf. figure 2) encapsulating methods and data needed for fulfilling and moving around a certain task. This concept is called *Activity*.

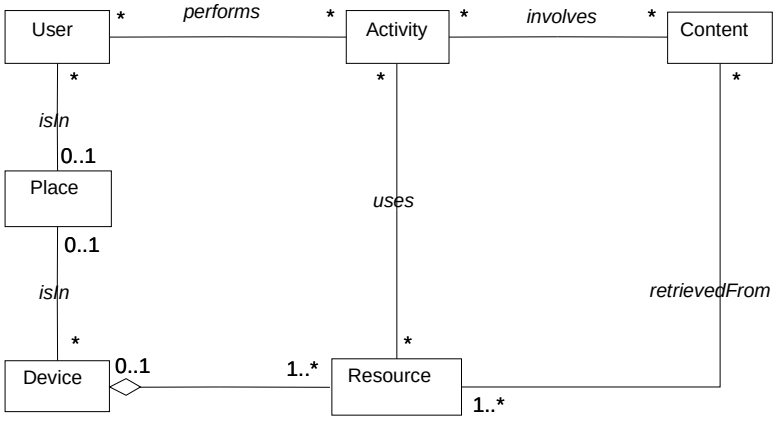

Figure 2 - The most important concepts from the domain model

An Activity represents a specific kind of 'task' a user wants to execute, for example watching a TV broadcast, browsing the web, watching a video or having a video communication. Each Activity contains information about the devices that are necessary for its task, methods for moving or copying it, and a way to get hold of a list of possible *Contents*. Content objects 'parameterize' Activities: While Activities specify the operations needed to execute a general task in an abstract manner, a Content object is needed to 'instantiate' this abstract description into the specific performance intended by the user. Examples for Content are (in the same order as above): different TV channels, selected URLs, movies in a video archive and other people's video phone addresses. Content objects are actually provided by Content Managers. They have the task of hiding the devices in the network, which provide the content. For this they retrieve content information from all eligible devices (e.g. all tuners or all VCRs) and merge them into one list.
In order to deal with locations, the concept of *Place* is introduced. A Place typically denotes a room but could also mean a part of a room. If the user wants to know what is going on in his system, he is shown Activities associated with Places. Therefore also any manipulation of Activities takes place on the basis of Places, not devices. An important form of such manipulation of Activities is relocating them. The *follow--me* functionality gives the user the possibility to take any Activity with him around the house to any Place that offers the technical preconditions for it (e.g. a screen for watching TV). The three different ways of relocating an Activity will be further discussed in section 4.2.
A more thorough discussion of concepts and realization of an easy to use UI is given in [2].
According to the domain model (cf. figure 2), the Activity, Content and Place abstractions shield the user from being actually concerned with devices. All this allows the user to think about what he wants to do and where, instead of thinking about which devices he wants to use.

4. REALIZATION AND ARCHITECTURE OF WWICE

The concepts described in the last section have been realized in a prototype. The environment this prototype is aimed at can be seen in figure 3. Digital CE devices in several rooms are connected by 1394 to share functionality and content. Set Top Boxes (STB) or Digital TVs will execute the system software.

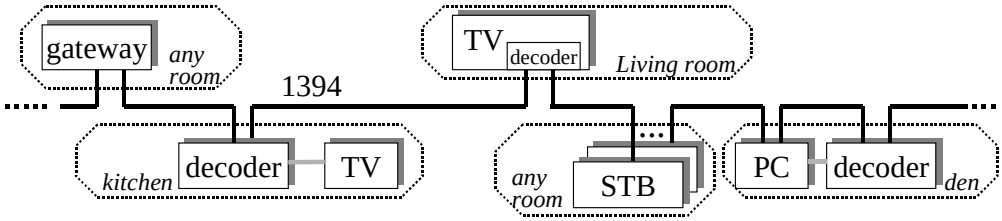

Figure 3: Devices in an IHDN environment

The overall goals when designing the architecture were easy extensibility and changeability. Changes arise naturally due to revisions of requirements and extensions were assumed to be necessary so the user can gradually increase the functionality of his IHDN and the supported hardware. Therefore a layered approach with a framework-like structure was designed. This structure leaves specific parts of the architecture open to extensions whereas others are claimed complete to ensure system stability. We will give a short overview over the different layers and will then more thoroughly describe one of the services.

4.1. Architectural overview

An overview of the WWICE architecture is given in figure 4. It is intended to run on any Audio/Video--Device providing the necessary processing power, e.g. STBs. To ease development, the functional prototype was built on standard PCs.
The *Platform* layer introduces abstractions for the used devices. The classes `Unit` and `Subunit` both provide standard methods for allocation, initialization and connection. Whereas a `Unit` is a physical device including connector, location etc., a `Subunit` is a functional component within a device. For each new hardware to be integrated into the system a subclass of these has to be included. It is intended that most of the standard methods do not need to be overwritten.
The *Middleware* layer provides all functionality that is necessary to allow easy realization of applications which follow the user interface concepts of section 3. The `Registry` contains static information about the system and its environment such as references to the other services and the `Units`/`Subunits` available in the system. The `Resource Managers` control the allocation of local and remote `Subunits` and network bandwidth. The `Graph Mapper` (cf. section 4.2.) maps abstract representations (i.e. graphs) of device networks needed by Activities onto the devices available in the system. This mapping process allows Activities to concentrate on the service they provide to the user instead of on how to find, allocate and connect devices. The `Factories` are used to create remote instances of a class and the `BootManagers` control the start-up phase of the system. None of these services needs extensions when adding new user-functionality or hardware.

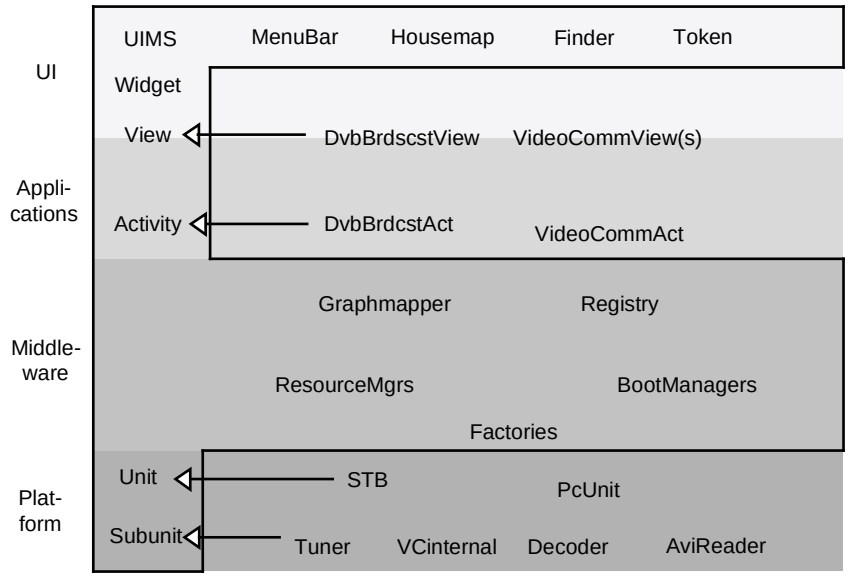

Figure 4: The WWICE software architecture

In the *Application* layer the implementation of all user-functionality is located. For each new functionality mainly three classes have to be inserted: `Activity`, `View` and `Activity-Icon`. The `Activity` relates to the Activity concept mentioned above. An `Activity` has to provide by default methods for initializing, starting, stopping, moving, copying and extending itself. Some of the methods have to be implemented anew for each `Activity`, others can be inherited from the base class with only minor modifications. Since these methods are declared in the `IActivity` interface, each Activity can be controlled by the UI-components independent of the particular `Activity` implementation behind it.

A `View` provides a GUI to its related `Activity` in accordance with the Model-View-Controller Pattern[10.] where the `Activity` is the Model and the View controls the UI elements provided by the User Interface Management System (`UIMS`). The ActivityIcon (not shown in the architecture) provides another View of the Activity. It displays an icon and a menu for the `Activity` and thereby allows more specific manipulations of the `Activity` using the UI-applications `MenuBar` and `Housemap` (cf. figure 5).

The *UI* layer is again a complete layer. It consists of the `UIMS` and several UI-'applications'. The `UIMS` provides basic GUI-elements and control over overlay and PiP-functionality. The UI-applications can be considered special management applications that – in contrast to the Activities – are not concerned with any Content but with the Activities themselves. The `Finder` lists kinds of Activities and related Content and allows the user to start them. The `MenuBar` gives an overview of the local Activities and provides control over their appearance.

Finally, the `Housemap` (cf. figure 5) shows the Activities in the whole house grouped according to the defined Places. This allows movement of Activities and also some remote control. In addition to the graphical UI also other forms of interaction exist. As an example, the class `Token` relates to a small physical object the user can take with him as a representation of an Activity. This allows the user to handle Activity movement in an even easier manner. Our experiences with these tokens together with those from seamlessly integrating speech control illustrated the benefits of a flexible UI layer, with extensibility similar to that of the Activity layer.

Figure 5: Menubar (left) and housemap (right) of the WWICE system

The architecture we choose provides extensibility coupled with uniform user interaction. New Activities can be integrated by implementing appropriate subclasses of `Activity`, `View` and `ActivityIcon`. Thus, a new Activity always provides the necessary interfaces and can therefore be controlled by the standard UI-applications known to the users. Also the results of basic user interactions are independent of Activity implementation because all Activities use the same middleware services to realize them. By providing more powerful services and richer interfaces we will increase the advantages of this architecture for the user as well as for the application developer.

4.2. The Graph Mapper

For the processing and presentation of multimedia data we use the notion of Graphs[8]. For each type of Activity, a Graph is defined. Its nodes (vertices) represent entities that control and process data, while the edges represent data connections between these entities. An Activity Graph is directed and has one or more source nodes (ones that only have edges pointing away from them) and one or more sink nodes (ones that only have edges pointing towards them).

The granularity that is used in the Graph of an Activity in such an approach is arbitrary. In our system, we have chosen a rather coarse-grained representation, where the nodes typically correspond to Subunits. For example, television watching requires a Graph consisting of a tuner, a demultiplexer, a decoder, and a screen. These Subunits have to be connected to each other, with connections leading from the tuner (source node) to the demultiplexer, from the demultiplexer to the decoder and from the decoder to the screen (sink node). The connection between Subunits may be either within a Unit or via the network.

In order to actually run an Activity, the Graph of that Activity has to be mapped to physical Subunits and network connections. This means the Graph Mapper has to find devices that are Subunits of the type requested by the Activity Graph, are currently available, i.e. not in use by another Activity, and fulfill any further restrictions put up by the Activity Graph.

In order to find the requested Subunits, the Graph Mapper contacts the registry to find out which Subunits exist in the system. It then contacts the overall Resource Manager to know whether these Subunits are available. The Graph Mapper also contacts the network to get a free channel, and asks the Units to connect the Subunits to each other or to the network. Possible restrictions when choosing Subunits will be discussed in the following paragraph.

Criteria for choosing a device

A Graph Mapper can allow for a wide variety of criteria to be passed along with the request to find a device to map a node onto. Simple examples of such criteria are:

- **Location:** Some Subunits (e.g. a tuner) can be equally used regardless of their location, while those that interact directly with the user (screen, camera, microphone, speakers) have to be in the desired Place.
- **Content:** Not all instances of a Subunit are able to provide the same content, e.g. a satellite tuner offers different channels than a cable tuner.
- **Quality of Service:** Different instances of a Subunit may provide different levels of quality, e.g. a camera offers a certain resolution and frame rate.

During the mapping process, other restrictions can occur, influencing the choice of devices:

- **Supported Stream Formats:** Audio and video streams can be provided in different formats and obviously combining different devices only makes sense if they support the same format or if a transcoder Subunit can automatically be included.
- **Network Capacity:** If there is not enough bandwidth available in the network, either a different device has to be chosen (e.g. one that is locally connected to the other within a Unit), or a different stream format with less bandwidth demand has to be used.

In the current implementation, the choice of location is already realized, while the other will be addressed in the next versions. To enable this location handling, the concept of *Subunit Sets* is introduced. A Subunit Set is a representation of Subunits that are located in the same Place. It is hence possible to request a Subunit of a certain type from a Subunit Set (e.g. a screen within the living room).

Relocating Activities

We will now briefly discuss how relocating Activities (the follow me concept) can be handled by the Graph Mapper.

If a user wants to join an Activity in another place, this can be modeled by extending the Graph. For example, if dad (in the bedroom) wants to join his son (in the living) with watching TV, the Graph can be extended by adding another decoder and screen, while keeping only one tuner and demultiplexer. This results in the Graph now having two sink nodes.

If a user wants to copy an Activity (i.e. join what the other family member is doing right now, but then have independent control), a new Activity based on the same Graph has to be started. In the example above, another tuner and demultiplexer would be allocated as well as another screen and decoder. This results in two Graphs, each with one source and one sink.

For moving an Activity, the sink node (which is usually defined as one specific device, e.g. the living room screen) has to be replaced by another one (e.g. the kitchen screen), which has to be connected to the source and those intermediate nodes which can remain the same. The original sink node can then be released. It is also possible that some of the intermediate nodes (or even the source node) are also relocated, e.g. using the kitchen decoder rather than combining the living room one with the kitchen screen.

Implications for Activity Developers

The concept of Activity Graphs that are mapped onto physical Subunits and network connections allows for a very general and simple interface that can be offered to Activity developers. Without the Graph Mapper, a developer would have to provide all the functionality needed for initializing, moving, extending and destroying the Activity. By using the Graph Mapper, the Activity only has to define the Graph it needs for starting up and what parts of the Graph should be moved in case of a follow me operation respectively duplicated in case of an extend operation.

5. SUMMARY AND OUTLOOK

The upcoming In-Home Digital Network leads to new distributed multimedia system solutions for the home environment. Several standardization and cooperation initiatives are currently working on the definition of technologies and interfaces for networking, protocols, and distributed system solutions in order to provide guaranteed interoperability of products from different vendors.

In addition to the interworking of devices, the WWICE system, being presented in this paper, focuses on an overall system environment for IHDN applications. Thus, it is an architecture for networked digital audio / video systems in the home. High-level requirements for such a system, derived from scenarios, are "Network and Device Transparency", "Location independence" and "Uniform way of handling". Our domain model covers these requirements, especially by the concept of Activity, allowing an abstract representation of all In-Home applications. Thus, we achieve a coherent control, even for different types of applications, and support moving, extending and copying them.

Its layered software architecture realizes WWICE as an efficient distributed multimedia system. The architecture allows for easy extensibility and changeability, which is required to support additional hardware in this new technology area as well as to introduce new applications quickly and consistently. We particularly focused on the middleware, and one of its services, the Graph Mapper. The Graph Mapper supports the flexible set-up of topologies of processing units and communication units, which are required for each Activity. That way, we achieve a very simple and general interface for Activity development.

Enhancement of the Graph Mapper will widen its functionality and the supported criteria for selection of Subunits. With these and further changes we will continue to explore concepts necessary for the digital home of the future.

REFERENCES

1. Philips, Sony, Matsushita, Thomson, Hitachi, Toshiba, Sharp, Grundig. The HAVi architecture – specification of the home audio / video interoperability (HAVi) architecture. Version 1.0 beta, October 1999.
2. Richard v.d. Sluis. Interfacing users to an IHDN system – experiences of the WWICE project. Submitted to CHI 2000.
3. IEEE. IEEE standard for a high performance serial bus. Standard 1394, IEEE, 1995.
4. IEEE. P1394b – draft standard for a high performance serial bus. Standard Draft 0.80, IEEE, August 1999.
5. G. Luetteke. The Multimedia Home Platform, in: http://www.dvb.org/dvb_articles/dvb_mhp98cm.htm
6. Joel A. DiGirolamo, Richard Humpleman. The VESA home networking initiative. White Paper release 3. VESA Home Network Committee, 1999.
7. Burkhard Stiller, Christina Class, Marcel Waldvogel, Germano Caronni, Daniel Bauer. A Flexible Middleware for Multimedia Communication: Design, Implementation, and Experience. IEEE Journal on selected areas in communications, Vol. 17, No. 9, IEEE, September 1999.
8. Kurt Rothermel, Ingo Barth, Tobias Helbig. CINEMA: an architecture for configurable distributed multimedia applications. In O. Spaniol, A. Danthine, W. Effelsberg, editors, Architecture and Protocols for High-Speed Networks, pages 253-271. Kluwer Academic Publisher, 1994.
9. David Duk, Ivan Herman. A standard for Multimedia Middleware, in W. Effelsberg: *Electronic Proceeding of the 6th ACM International Multimedia Conference (MM98)*. ACM, Bristol, UK, September 1998 http://www.acm.org/sigs/sigmm/MM98/electronic_proceedings/duke/index.html
10. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns. Addison Wesley, Reading, MA, 1995.