

Context-Aware Middleware for Mobile Multimedia Applications

Oleg Davidyuk, Jukka Rieki, Ville-Mikko Rautio and Junzhao Sun
Department of Electrical and Information Engineering
P.O. BOX 4500, University of Oulu
Finland, 90014
358 8 553 2544

{oleg.davidyuk, jukka.rieki, ville-mikko.rautio, junzhao.sun}@ee.oulu.fi

ABSTRACT

We present a context-aware middleware for mobile multimedia applications. The middleware offers functionality for service discovery, asynchronous messaging, publish/subscribe event management, storing and management of context information, building the user interface, and handling the local and network resources. It supports a wide range of context information including location, time, and user's preferences. Further, it allows controlling the connectivity of the device; the middleware is capable of switching traffic from one network connection to another. It can locate the services and software components as well. It facilitates development of multimedia applications by handling such functions as capture and rendering, storing, retrieving and adapting of media content to various mobile devices. The middleware offers facilities for media alerts, which are multimedia messages that are generated when predefined context is recognized. With these capabilities, it enables development of complex context-aware multimedia applications for mobile devices.

Keywords

Multimedia content adaptation, context-aware and mobile computing

1. INTRODUCTION

Personal digital assistants (PDAs) and mobile phones nowadays offer a versatile set of facilities, which enables developing various new applications. Context-aware mobile multimedia is an application area that has been attracting the attention of researchers in recent years. This application area offers a big business potential, as the users of mobile devices want to access multimedia content and context-aware services from their devices.

However, mobile devices have limited bandwidth, small screen size and low computational power. Hence, the multimedia content has to be adapted to fit the constraints of the device.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MUM 2004, October 27-29, 2004, College Park, Maryland, USA.
Copyright 2004 ACM 1-58113-981-0/04/10...\$5.00.

Mobility and context-awareness introduce further challenges. The applications have to adapt themselves to a changing environment, i.e. the recognized context.

An intermediate software layer, which performs the tasks related to mobility and context-awareness and provides multimedia support could solve the problems raised. It helps to avoid the increasing complexity of the applications and lets the developers concentrate on the application-specific tasks.

Several middleware solutions have been developed in the area of distributed computing, such as CORBA [1] and J2EE [2]. However, these technologies are intended to be used in networks with fixed infrastructure and require the devices to have a lot of available resources. Hence, they are not suitable for mobile computing. Furthermore, they have no support for multimedia.

The Service-oriented Context-Aware Middleware (SCAM) project [3] provides an architecture for context-aware mobile services. It concentrates on context-related facilities and has few services to support mobility of the applications. It does not offer functionality for multimedia applications, though. The CAMPUS project [4] presents a middleware for context-aware applications. Its novelty is in the context model, which is based on the fuzzy sets theory. However, the middleware is suitable only for fixed devices, because it has no support for mobile and multimedia applications.

The CORTEX architecture [5] is a middleware for context-aware distributed applications. It has functionality for messaging, service discovery and resource management. The system is not capable of controlling the network resources and provides no adaptable UI for applications. Further, it does not support multimedia applications. The DISCWorld system [6] concentrates on mobile robot systems. Its middleware includes several components providing functionality for communication. The aim of the DISCWorld middleware is to solve the network challenges of connecting a set of robots and controlling PDAs into one network, hence the middleware focuses on messaging facilities only. Furthermore, it has no support for multimedia applications.

The middleware service for distributed multimedia applications, offered by Lohse et al. [7] provides functionality for rendering audio and video in distributed environment. It also supports adaptation of multimedia content for restricted resources of mobile devices. However, the middleware solution does not offer functionality for context-aware applications.

The QoS DREAM framework [8] is focused on supporting the development of context-aware multimedia applications. It has event messaging component, data storage and distributed multimedia service. The framework supports streaming media, but

the context is presented by location information only. The framework does not support mobility of the applications (e.g. the applications cannot migrate from one host to another). Besides, it assumes that the network connection is fixed. The mobile devices are used as location sensors only.

Lau and Lum [9] developed a decision engine for multimedia content adaptation. They offer a real-time processing system for rich multimedia elements, which allows accessing the content using PDAs. Lemlouma and Layaida [10] offer a framework for adaptation of content delivery to mobile devices. However, these solutions do not offer functionality for the development of mobile context-aware applications, even though the projects are focused on the delivery of multimedia to mobile devices with constrained resources.

From this short survey, it can be noticed that there is a gap between context-aware solutions for mobile applications and solutions for distributed multimedia applications. The first ones handle contextual information and support mobile applications by hiding heterogeneity of the hardware. They offer solutions that are light enough to be executed in a mobile environment where the resources are limited. The solutions for distributed multimedia applications assume that mobile devices have restricted resources and, hence, offer functionality for multimedia content adaptation. Besides, they support mobile applications by providing network transparency, making them able to operate in a distributed environment. These middleware solutions are not able to perform context-related tasks, however. The only framework that supports context-aware, distributed and multimedia applications is QoS DREAM. However, it has a number of constraints, which were described above.

Furthermore, neither context-aware middleware solutions for mobile applications nor solutions for distributed multimedia applications are capable of controlling network resources at the middleware level.

In this work, we present a middleware solution for development of context-aware and mobile multimedia applications. The middleware is able to perform most of the tasks dealing with context-awareness and mobility, and it focuses on multimedia support. It locates the services and components, provides asynchronous messaging between the applications, stores and processes the context data collected from various sources, manages local and network resources, etc. The support for multimedia includes functionality for capturing, rendering, processing, adaptation and storing the media content. The middleware introduces a new type of events, called media alerts, and facilitates building media messaging applications. Media alerts are multimedia messages that are generated when a predefined context is recognized.

This paper is organized as follows. Section two specifies our context model and context-aware middleware. We describe the functionality and architecture of the CAPNET middleware in the third section. The implementation and the prototype are presented in the fourth section. The fifth section contains a discussion and summary of the work.

2. ARCHITECTURE REQUIREMENTS

Context awareness of ubiquitous mobile applications has recently been attracting the attention of many researchers as an interesting and perspective topic for research. Context awareness is an essential feature of mobile systems, because almost all the

ubiquitous applications utilize context information in their operation. We adopt Dey's definition of context as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves" [11]. An example of context information can be a user's location, time, user's profile, local resources of the mobile device, available services, etc.

Context awareness characterizes a system to use context information when it performs its tasks. In the present paper, we use the task-oriented definition of context awareness: "a context-aware system uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." [11]. That is, the main stress is put to the context that is relevant to the task.

Context awareness involves performing data acquisition from sensors, context recognition and other tasks necessary to complete before the context can actually be used. Delegating the data acquisition and context processing tasks to applications makes them almost impossible to reuse. One solution to such a problem is to decouple the tasks from applications and move desired functionality to the lower layers. Such layers, which serve the needs of applications, usually form a special layer called middleware. We keep to the definition of middleware given by Couluoris as a "software layer that provides programming abstraction as well as masking heterogeneity of the underlying networks, hardware, operating systems and programming languages" [12]. This middleware layer hides the heterogeneity and distributed nature of devices measuring the context information. A context-aware middleware serves the context needs of applications. The functionality of such a layer is discussed in detail in the next section.

A context-aware middleware has to provide the applications with the following context-oriented functionality [13]:

- support of a variety of sensor devices,
- support of the distributed nature of context information, because the data comes from different sources,
- providing for transparent interpretation of applications and abstraction of context data,
- maintenance of context storage, and
- control of the context data flow.

Mobility introduces a number of constraints to the middleware[14]:

- the bandwidth is low and hosts can be unreachable due to network partitions or poor coverage,
- the local resources like memory capacity and CPU power of the device are very limited,
- the communication between the system components is asynchronous, and
- the execution environment is dynamic.

Dynamic execution environment refers to constantly changing distributed mobile systems. Hosts are not statically connected to the network and the configuration of such systems is hard to predict. Therefore, the middleware must support discovery of hosts and service in such a continuously changing system.

Furthermore, multimedia applications set requirements for communication and computation. Multimedia algorithms need a

lot of bandwidth and processing power. Hence, the middleware has to be capable of:

- Using the available bandwidth: the middleware has to use all the different, available connections and switch to the connection that best fulfills the requirements at the given moment;
- Controlling the place of computation: to cope with limited resources the middleware needs to decide where the computations should be performed based on the situation at hand.

Furthermore, the middleware must support various multimedia devices such as video cameras, microphones, etc. Finally, there are requirements that the middleware has to fulfill to make the system adaptable [15]:

- Triggering of adaptation on a system-wide level,
- Support for system-wide adaptation policies, and
- Providing a common interface between devices and middleware.

3. CAPNET MIDDLEWARE

CAPNET middleware has been developed to fulfill the requirements set to the architecture by context-awareness, mobility of the software components, multimedia applications and adaptation.

The CAPNET middleware is located below the application layer and on top of the layer of existing technologies. All the functionality provided by CAPNET middleware is divided into a set of components. Each component offers a particular service to other components and applications. The CAPNET middleware includes such components as connectivity management, service discovery, messaging, component management, user interface, context, media and context-based storage. The structure of the CAPNET middleware is shown in Figure 1.

CAPNET middleware fulfills all the requirements set by context-awareness. It supports different devices to collect the location data and record stream video. The data comes from distributed software components, located physically at different networked devices. Context abstraction is performed transparently through intermediate components, so applications receive already abstracted data, instead of rough measurements. The middleware offers a full set of context storage facilities, like storing any context-related information, even including files.

The mobility of the CAPNET system is supported by light component framework. The components are executed at the device that has enough resources. The connectivity management component controls the bandwidth and overall traffic. The messaging component, service discovery and component management provide the necessary level of transparency for applications making them able to operate in mobile environment.

The requirements set by multimedia applications are also fulfilled with the component framework and the basic services that it offers.

The adaptation is performed at middleware level, which facilitates easy building of multimedia applications. Every application defines its resource and network requirements, which have to be taken into account during the execution of the application. Besides, applications define network policies that constrain connections and traffic. Finally, the middleware

components have common interfaces to the layers located beneath. It allows easily extending the set of the hardware supported by CAPNET middleware.

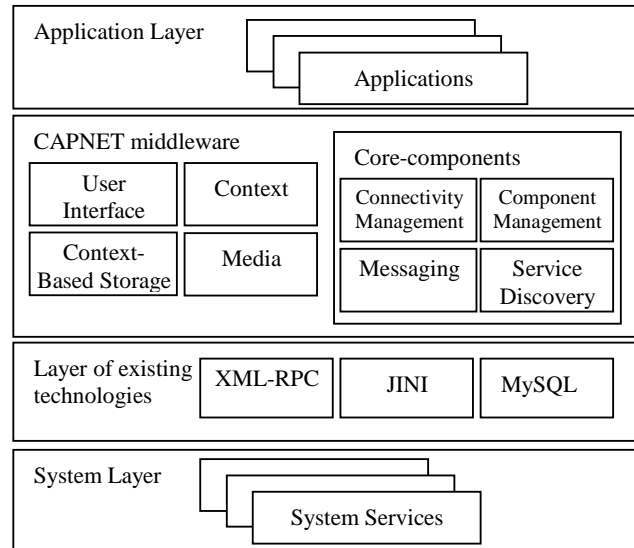


Figure 1. The architecture of CAPNET middleware.

3.1 CAPNET Components

The CAPNET middleware components are atomic software entities with explicitly defined interfaces. The internal structure and complexity of the implementation are hidden behind the component's interface. All the components are represented with proxies or stub components. This proxy approach helps to achieve the distribution transparency in the environment. Hence, the components communicate with each other in the same fashion regardless of their real location – local or remote. The components can be searched and started at system startup or on the fly, while applications are running. This facilitates the adaptation of the middleware to the environment.

The CAPNET core contains component management, messaging and service discovery components. They provide a full set of services required to support the other components and applications in a distributed environment.

Component management performs the controlling of the components and their stubs. It starts the components at startup, initializes and creates proxies, processes the requests to gain access of the components to each other. When a handle to a component is requested, the component management returns the reference to the proxy component, instead of reference to the real component. If the required component is not available at the local device, the component management loads it from the Internet from a specified location. The component management is able to make decisions and move components to the remote host if the device does not have enough local resources (e.g. memory or CPU power is limited) to execute the component. This is required when the media components need to perform processing or adaptation of the multimedia content.

The communication in CAPNET middleware is the responsibility of the messaging component. It creates channels and offers the functionality for asynchronous communication,

channel-related operations and supports remote procedure calls in the environment. A channel is a logical link between different physical applications, which are located at networked devices. The messaging component provides the location transparency together with component management.

The event-notification mechanism is based on a publish-and-subscribe model. The components and applications (event consumers) can subscribe events of a specific type from event producer components.

In the dynamic distributed environment, the availability of services and hosts changes in continuous fashion due to the network disconnections and poor coverage in some locations. Hence, service lookup becomes an important issue, because there is no centralized server and the default server does not exist. Location transparency is achieved with service discovery facilities. This component locates services and available components. Clients request service discovery to find services provided by other components. Service discovery looks up and returns the list of the matches to the client, which selects the desired component from the list and requests component management to get reference to the selected component.

The connectivity management and media components are described in more detail in the following subsections. The rest of the middleware components are optional to be running in the device, but necessary to support context-aware mobile applications. Among service-enabling components are context-based storage, context, and user interface components.

The storage facilities in CAPNET middleware are provided by context-based storage. It mainly stores and retrieves data by request. The component deals with synchronization of the data and alerts the clients when the content of the database is changed. The context-based storage is responsible for defining the privacy and access rights to the user's data.

The responsibility of the context component is providing the context information: context delivery, processing and service. The context component plays the role of a wrapper for context sensors. Furthermore, it is used as a server component, when the context data comes from distributed sources. The context component along with context-based storage offers context history maintenance facilities.

The user interface (UI) component acts as a UI front-end to a mobile device and supports the design and implementation of application UIs using three different techniques: abstract UIs (XML), plug-in UIs (downloadable Java code) and Web-based UIs (HTML). The UI software developer may choose the technique that best suits her needs, but the UI has to be separated from the application logic. For abstract UIs, the component presents a simple interface for building and modifying the UI implementation as well as exchanging messages and events between the application and the UI implementation. In case a Java plug-in UI is used, only message exchange is supported. Messages are delivered using the CAPNET messaging mechanism. Web-based UIs are launched in the Web browser of the target device and communication between the application and its UI has to be built separately using the Web techniques (Web server, CGI-scripts, Java Server Pages, etc.).

3.2 Connectivity

The connectivity management component controls and monitors the connections of the mobile device. The component hides the

details of the actual connections behind channels. Channels provide network transparency for the application layer and allow dynamic adaptation of the network facilities without interfering with ongoing communication. That is, components use channels and they do not see which connection is used to transfer the data. The connection can even be switched in the middle of communication without the components noticing it. Currently, connectivity management supports the TCP/UDP, HTTP, Multicast and TCP listening channels.

The clients can request real-time information about the status of the channel, e.g. bandwidth and availability of the connection. Besides direct requests to the component, the clients are able to subscribe this information and get notifications when the status of a channel changes (for example a connection is terminated or becomes available).

The connectivity management component controls the channels by switching the connections on the fly (e.g. from WLAN to GPRS) according to the defined policies. The channel policies can be changed through the public interface on the fly by the user or an application. We are not aware of any middleware solutions that can control connectivity at the channel level, limit the overall traffic or bandwidth and dynamically switch connections while clients are using them.

The connectivity management has three subcomponents: connection controller, policy manager and connection monitor. The architecture of the connectivity management component is shown in Figure 2.

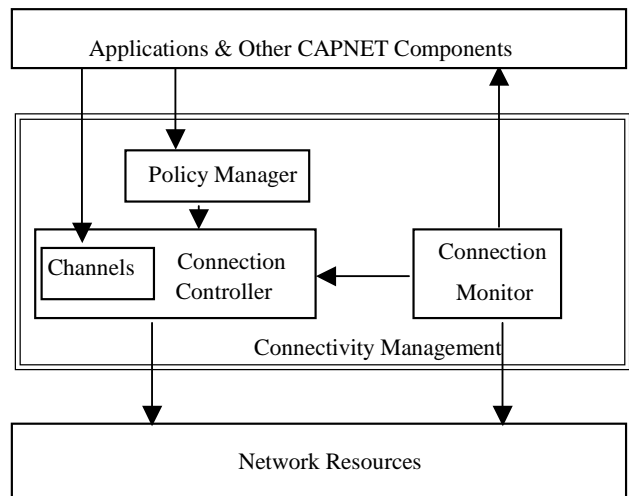


Figure 2. The architecture of the connectivity management component.

The connection monitor senses the channels' context information. It collects both QoS parameters of channels (e.g., round trip time, RTT) and network characteristics like IP addresses of the local and remote hosts. The policy manager is responsible for defining criteria and rules for channels. The policy manager allows binding the rules to the particular channel or device. The connection controller is capable of maintaining both the policy manager and connection monitor subcomponents. Besides that, the connection controller manages the channels as the client desires.

3.3 Media Components

The media components facilitate portability and scalability of native media capabilities across the various devices. The goal is to provide developers of media applications with functionality to capture images, audio and video. Media can be processed and stored either locally or remotely. Media content can be adapted for various devices. The adaptation can be performed both at a local or a remote device.

Media components are divided into core and optional components. The core media component is static and it is always present when media functionality is used. The optional, dynamic components are loaded by request, depending on the needs of applications and available resources at the device. So, if the device does not have enough resources, the dynamic media components can be started remotely, at the server side. The dynamic components can be constrained to run only at the server side. The architecture of media components is shown in Figure 3.

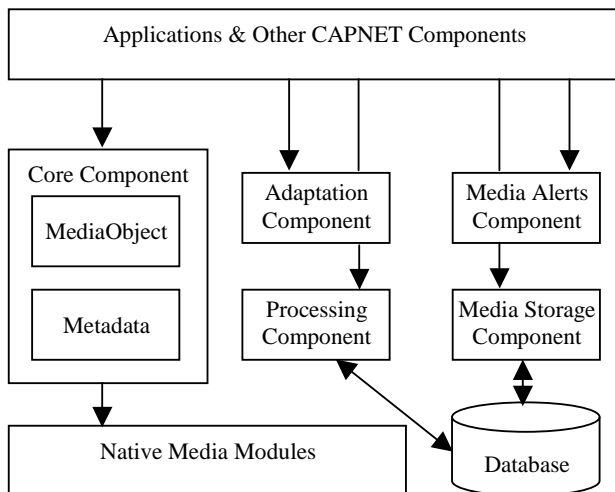


Figure 3. The architecture of media components.

The core component creates and performs serialization of media objects and handles related metadata. The media object contains the data of all media sub-elements, relations of media elements and metadata of each sub-element. The component supports three types of relations between media elements: spatial, temporal and navigational relations. Spatial relations define how the elements are spatially related to each other. The temporal relations define how the media elements are temporally related, e.g. element B has to be played before element A. The navigational relations define the links of elements to each other, e.g. element A can contain a link, which requires the media playback to play element B. The spatial, temporal and navigational relations are presented accordingly to the SMIL 2.0 [16] standard.

The rest of the components are dynamic and they offer functionality to store and retrieve media elements and process them if necessary. The components are also capable of capturing and rendering media objects. Furthermore, a media alert component offers an interface, which supports a push-type delivery mechanism of media objects. Media storage component is an interface between the database and media components. It

uses metadata to create indexes and offers a flexible way to query the database for media objects.

Applications require adaptation of media objects to make them suitable for playback on different devices. This functionality is supported by media components as well. The clients pass the media objects and information of the terminal device to the media components. The dynamic components retrieve the facilities of the device from the database and adapt the media object according to these constraints. The adaptation is performed either locally or remotely (e.g. at server side).

Capture and rendering facilities are supported by media components to provide an easy way to create media objects and play them. Before the actual capture or rendering starts, the media components check if the target device has enough resources to play or capture media. Instead of CAPNET media components, it is possible to use external, third party components to perform capture or rendering of media objects.

Media alerts are events associated with media content. They are generated when some media-related context occurs, e.g. a camera detects movement or a microphone detects speech. Components and applications can subscribe to receiving the alerts that they are interested in from the media alert component. Besides, the media alert component delivers media alerts between other components and applications by request.

The media components support also invocation of the media alerts by applications. So every application can invoke its own kind of alerts. This allows creating a flexible infrastructure for delivering media messages. The media alerts are delivered in an asynchronous way. If the client is not available to receive a media alert, it will be queued and delivered later, when the client is available.

4. PROTOTYPE AND EXPERIMENTS

Two prototypes have been implemented to verify the CAPNET architecture. The prototypes were tested with Compaq iPAQ PDA equipped with a WLAN network card. The programming environment of the device includes Insignia Jeode Java virtual machine. The devices are located by Ekahau WLAN positioning engine. It locates the devices according to the measured strength of the WLAN signal. The prototype utilizes a MySQL database, which is running at the server side. All the communication between components is performed by using open-source Marquee XML-RPC library. The service discovery is based on Jini.

The functionality of the prototype is implemented as described below. The components can be searched, downloaded and started dynamically. The requests of the clients to get references to other components are processed by component management. It is capable to start components both at local and remote hosts. If a component is not available at the required host, the component management is able to download it from the Internet. The capabilities and requirements of the components are presented in the form of XML descriptions. Such descriptions include both optional and mandatory characteristics of the components, like names of the component's interfaces, unique ID numbers, URLs of the component's code and so on. The descriptions are used by component management and service discovery components. The messaging component is able to perform remote procedure calls between components regardless of their location – be it at the same or different hosts. The functionality implemented in the messaging component includes

sending of asynchronous messages. All the message-based communication in our prototype is based on XML-RPC protocol. We selected XML-RPC because it is a lightweight protocol and its learning curve is shorter than CORBA's IIOP or SOAP. The disadvantages of XML-RPC are that it does not allow sending objects as pass parameters of functions and it dictates to use unnamed, positioned structures in XML messages [17]. The mobility limits the network resources in our prototype; therefore we are not able to utilize the whole features offered by object-oriented XML-RPC.

We implemented our own component approach, because CORBA and J2EE are suitable only for fixed distributed solutions, not for mobile web applications. However, the CORBA implementation works faster than XML-RPC based client [18]. At the same time, both CORBA and J2EE require more time for learning and more bandwidth for normal performance.

The connectivity management component is capable of changing the connections from one type to another without interference to the work of applications. This channel adaptation is transparent and does not require any decision-making process from applications. Applications and clients just need to define the channel policies, so the connectivity management changes the connection accordingly to these policies.

The service discovery component is capable of locating services and other components using Jini and an internal database. When needed, the clients ask service discovery to look a service or component up. The service discovery provides the client with the list of matches. Then, the client selects one entry from the list and requests the component management to get a handle to the desired component. The service discovery performs location-aware discovery of the services as well. For example, the clients can constrain service discovery to request a list of services available near the location of the client. The performance of the dynamic service discovery was tested as well.

The CAPNET environment supports different kind of events. The context-related events are produced and collected in a push model. Whenever the event occurs it is forwarded to the context component, which sends it to the component responsible for that event. The events are delivered in a publish-and-subscribe fashion. The event suppliers have to register their interest in some event type from a certain component event producer. The middleware supports internal events, which are delivered directly from one component to another. The internal events are the responsibility of the messaging component.

The media component is capable of capturing the video and taking images. The synchronization of audio and video streams has not yet been completed, so the video is recorded and played separately from the sound. The media component is capable of creating media objects and handling them related to the objects' metadata. The rendering functionality is implemented, as well as media alerts and storage facilities. The adaptation of media objects is at an early stage of implementation. The other components, like context, UI and context-based storage are implemented as it was described in the previous section.

We experimented with prototypes for routine learning and business meetings. The first prototype is capable of learning user's habits and able to support location-based reminders. It consists of four applications: profile manager, reminder, calendar and personal assistant. The calendar application is used to store appointments and meetings. The context component gets events from the calendar via the reminder application. Finally, the profile

manager detects the routines from the context component. The personal assistant application is used to browse the applications at the mobile device.

According to the prototype scenario, the user buys his phone and starts using it. It learns that the user has a habit of switching the phone on the silent mode during weekly meetings. After it has detected a routine, the system offers to switch the phone on silent mode automatically if the user has a meeting. Furthermore, the prototype supports location-based reminders. So, the user attaches a reminder to a location and then the system recognizes the place and notifies the user by popping up a message.

The prototype for business meetings is based on a scenario as follows. The user has to make a presentation. He remotely checks the available services in the meeting room. After the presentation is created, he adds the presentation file to the event in his calendar application. In the morning the user gets a reminder about the meeting. When it is his turn, the user searches for available services in the meeting room, and then selects the slide projector. The list of files associated with the meeting is shown as well. The user browses the required file and starts the slideshow by using the mobile device as a remote control unit. During a meeting the user's wife tries to reach him by phone. The wife gets a notification that her husband is at the meeting. She decides to leave a video message. When the meeting ends, the user gets notification about the video message and watches the message.

The prototype for business meetings needs phonebook, video messaging, projector, and service browser and file browser applications. The video messaging application is capable of recording and showing the video messages. The projector application supports controlling the actual projector via mobile device. The service browser is the application that helps the user to request service discovery for available services. The file browser application is an interface to the database. The scenario for the media component is presented in Figure 4.

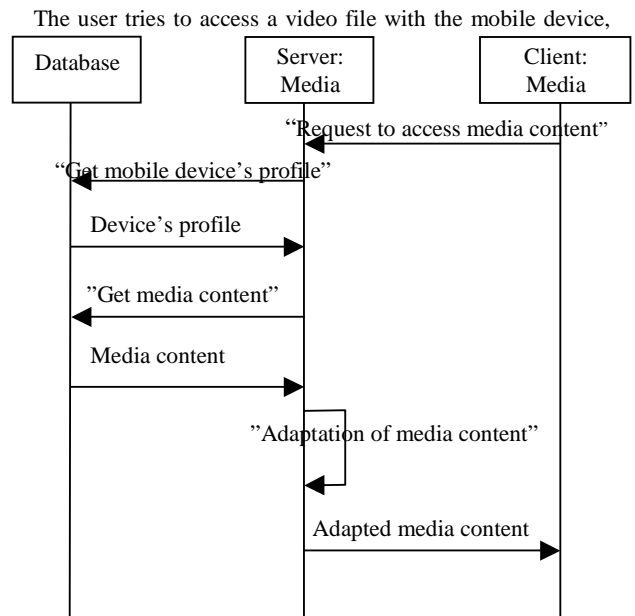


Figure 4. Media components scenario.

but it is not capable of playing the file, because of the small

screen. The media component, located at the mobile device, requests the server media component, located at the server, to get the video file. The server component, in its turn, queries the device profile from the database and then retrieves the original video. After the adaptation is completed, the server sends the file to the mobile device, where the user can play it. In the experiments, both the prototypes worked as expected. The media scenario was performed as explained. The media components processed requests to provide access to the video clip. They retrieved the device profile from the database and delivered the video clip to the mobile device. The clip was rendered at the device. The adaptation mechanism of the media content is still at the implementation stage.

5. DISCUSSIONS

We have presented context-aware middleware for mobile multimedia applications. The middleware provides key features for both context-aware multimedia applications and their distributed environment. The middleware is decomposed into a number of components, offering certain functionality for applications. The middleware is developed to work in the mobile environment. Hence, the resources of the terminal devices are very limited.

The middleware is oriented to provide functionality for context-aware applications. It has a common interface for context sensors, so it is easy to embed a new sensor to the system. The sensors can reside on physically distributed devices. Hence, the context information is forwarded to a centralized context component that performs further context delivery and processing. Besides, the context component is responsible for context recognition. These context-related operations are performed at middleware layer, which makes context processing and recognition transparent for applications. The context-based storage is the component that provides data storing facilities. It is responsible for data synchronization as well. If the content of the database was changed, the component notifies the clients. They can subscribe their interest to receive such alerts from the context-based component. Furthermore, it together with the context component controls the context data flows in the environment.

The middleware supports component mobility; it can decide whether to start a component on the client side (mobile device) or server side (fixed PC) depending on the resource requirements of the starting application or component. The functionality of moving components during the execution has to be implemented in the next prototype. The distributed environment is supported by messaging, service discovery and component management components. The messaging component offers functionality for asynchronous communication and remote procedure calls. The service discovery component is capable of locating services, components and devices. The component management handles the components during their lifecycle. Furthermore, mobility of the hosts can cause disconnections and network partitions and thus makes prediction of disconnections an extremely difficult matter. The next prototype will use DB2 Everyplace engine as the database, which has embedded synchronization and backup mechanisms to prevent data loss in case of disconnections.

The middleware provides functionality for creating and handling multimedia content. It is capable of storage, retrieval, capture and rendering of media. The middleware offers a possibility for creation and invocation of media alerts that

facilitates development of multimedia messaging applications. Besides, it is capable of controlling the network resources and switching the channels according to the policies, which can be defined by multimedia applications. So the middleware is able to adapt to the situation.

A prototype of the CAPNET system has been implemented with the presented functionality. The application developers can use the services of the middleware and focus the design on application-specific issues. The CAPNET middleware is fully developed with Java, so it makes the prototype adaptable to mobile devices. Furthermore, the architecture of the middleware is built to increase the portability of the system to different platforms. The Java approach facilitates the interoperability of the implemented prototype, which is an important issue in the mobile environment. However, implementation of the system in the native language promises to be more effective than implementation in Java, because Java cannot access all the resources of the mobile device. The implemented media services enable rapid development of multimedia messaging applications by supporting a varied set of functions for multimedia creation, delivery, adaptation and processing.

Future work includes improving the architecture to get more reusable components. The middleware is decomposed into several sub-layers, and higher layers of the middleware can be easily ported to different OS. The messaging component in the next prototype is based on Session Initiation Protocol, which helps to solve the network partitioning problems. The component management takes more responsibility from the application layer and is able to move the components from one device to another during their execution with minimal interference to their work. The service discovery component is based on the RDF model. It offers an easy way to combine complex sequences of events and conditions to describe action-triggering rules. The context-based storage uses DB2 Everyplace as the database that definitely gives benefits when considering a dynamic mobile environment. It is capable of automatic data update and synchronization. The media component will be implemented to provide adaptation of media content as well as capture and rendering of video messages with sound.

Finally, we are implementing the prototype on Symbian OS. The basic functionality of core components like messaging and component management will be implemented on Symbian OS during 2004.

6. REFERENCES

- [1] Object Management Group: Common Object Request Broker Architecture: Core Specification, v. 3.0.3, 2004.
- [2] Sun Microsystems, Inc: Java 2 Platform Enterprise Edition Specification 1.4, 2004.
- [3] Gu, T., et al.: A Middleware for Building Context-Aware Mobile Services, *In Proceedings of IEEE Vehicular Technology Conference (VTC-Spring 2004)*, Milan, Italy, 2004.
- [4] Hisazumi, K., et al.: CAMPUS: A Context-Aware Middleware, *The 2nd CREST Workshop on Advanced Computing and Communicating Techniques for Wearable Information Playing*, Nara Institute of Science Technology, Nara, Japan, 2003.

- [5] Duran-Limon, H., et al.: Context-Aware Middleware for Pervasive and Ad Hoc Environments, Computing Department, Lancaster University, Bailrigg, Lancaster, UK, 2000.
- [6] Hawick, K.A., James, H.A.: Middleware for Context Sensitive Mobile Applications. *In Proceedings of workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing*, Adelaide, Australia, 2003, pp. 133-141.
- [7] Lohse M., Replinger M., Slusallek P.: Session Sharing as Middleware Service for Distributed Applications, *Interactive Multimedia on Next Generation Networks, Proceedings of First International Workshop on Multimedia Interactive Protocols and Systems (MIPS 2003)*, Naples, Italy, 2003.
- [8] Naguib, H., Coulouris, G., Mitchell, S.: Middleware support for context-aware multimedia applications, *In Proceedings of the 3rd IFIP WG 6.1. International Working Conference on Distributed Applications and Interoperable Systems (DAIS 2001)*, Krakow, Poland, 2001.
- [9] Lau, F., Lum, F.: A Context-Aware Decision Engine for Content Adaptation, *in Proceedings of the 8th annual international conference on Mobile computing and networking*, Atlanta, Georgia, USA, 2002.
- [10] Lemlouma, T., Layaida, N.: Adapted Content Delivery for Different Contexts, *In Proc. Of 2003 Symposium on Applications and the Internet*, IEEE, 2003.
- [11] Dey, A.: Providing Architectural Support for Building Context-Aware Applications, PhD thesis, Georgia Institute of Technology, 2000.
- [12] Coulouris; G., Dollimore, J., Kindberg, T.: *Distributed Systems Concepts and Design*, Addison-Wesley, 2001.
- [13] Dey, A., et al: The Context Toolkit: Aiding the Development of Context-Enabled Applications, *in Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems (CHI-99)*, Pittsburgh, Pennsylvania, USA, 1999.
- [14] Carpa L. et al.: Middleware for Mobile Computing, *In Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, Elmau, Germany, 2001.
- [15] Efstratiou, C., et al.: Architectural Requirements for the Effective Support of Adaptive Mobile Applications, *in Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware-2000)*, New York, USA, 2000.
- [16] W3C: Synchronized Multimedia Integration Language, <http://www.w3.org/TR/smil20/>, 2004.
- [17] Gabhart, K., Gordon, J.: Wireless Web Services with J2ME, *WebServices Journal*, Volume 2, Issue 2, 2002.
- [18] Gryazin, E., Burmakin, E., Tuominen, O.: Comparison of CORBA and Web Services middleware operation in wireless environments, *in Proceedings of International Workshop on Mobile Computing, (IMC 2003)*, Rostock, Germany, 2003.