CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Waseda University Repository

A Middleware Infrastructure for Building Mixed Reality Applications in Ubiquitous Computing Environments

Eiji TOKUNAGA Andrej van der Zee
Makoto KURAHASHI Masahiro NEMOTO
Tatsuo NAKAJIMA
Department of Information and Computer Science, Waseda University
3-4-1 Okubo Shinjuku Tokyo 169-8555, JAPAN
TEL&FAX:+81-3-5286-3185
{eitoku,andrej,mik,nemoto,tatsuo}@dcl.info.waseda.ac.jp

Abstract

Mixed reality is one of the most important techniques to achieve the vision of ubiquitous computing. Traditional middleware for mixed reality provide high level abstraction to hide complex algorithms for analyzing video images, but applications programmers still need to take into account distribution and automatic reconfiguration when developing mixed reality applications for ubiquitous computing.

Our middleware infrastructure hides all the complexities to build mixed reality applications for ubiquitous computing. Therefore, the development does not require advanced skills for ubiquitous computing. The paper describes the design and implementation of our infrastructure, and presents some scenarios and the current status showing its effectiveness.

1 Introduction

Mixed reality[3] is a promising technique for realizing the enhancement of our real world by superimposing computer generated graphics on video images. The technique is important in ubiquitous computing environments[1, 20, 25, 27] to enhance our real world by using information in cyber spaces. However, in ubiquitous computing environments, application programmers need to deal with ultra heterogeneity to support various devices and environments, and handling continuous media such as audio and video is very hard. Also, they need to take into account complex issues such as distribution and dynamic reconfiguration that increase development cost of continuous media ubiquitous computing applications. To solve the problem, it is important to provide a middleware infrastructure to hide the complexities to make it easy to develop the applications.

This paper describes the design and implementation of a software infrastructure for building mixed reality applications in ubiquitous computing environments. Traditional toolkits for mixed reality provide high level abstraction that makes it easy to build mixed reality applications, but application programmers still need to take into account distribution and automatic reconfiguration that make the development of applications very hard. The high level abstraction provided by our software infrastructure hides these complex issues from application programmers. Therefore, the cost to develop mixed reality applications will be reduced dramatically by using our software infrastructure. Although our paper focuses on how our system is used to build mixed reality applications for ubiquitous computing, our middleware can also be used to build many other ubiquitous computing applications that deal with continuous media.

The remainder of this paper is structured as follows. In Section 2, we describe related work and compare our framework characteristics with existing middleware. In Section 3, we show the design issues of our infrastructure. Section 4 presents the design and implementation of our middleware for distributed mixed reality. Section 5 presents two scenarios that show the effectiveness of our approach. In Section 6, we describe the current status of our system. In Section 7, we discuss strengths and weaknesses of our current design. We conclude the paper in Section 8.

2 Related Work

ARToolkit[2] is a software library that allows us to develop mixed reality applications easily. It provides several functions to detect square formed visual markers in a video frame and superimpose OpenGL based 3D Object on the markers in the video frame. ARToolkit is quite useful for Mixed Reality prototyping, but it does not provide distributed programming framework and heterogeneous device adaptation. We implemented continuous media components for mixed reality by reusing programs provided by the ARToolkit. Therefore, we can utilize most of ARToolkit functions in our distributed multimedia programming model and dynamic adaptation framework.

DWARF[4] is a component based framework for distributed mixed reality applications using CORBA. Our system also use CORBA for communication infrastructure. In this aspect, our framework is very similar to DWARF. However, our system is different from DWARF since our system offers automatic reconfiguration to develop mixed reality applications suitable for ubiquitous computing. It is very essential part of our framework because dynamic adaptation according to application context is one of the main issues in ubiquitous computing.

The VuSystem[16] is a framework for compute-intensive multimedia applications. It is divided into an in-band partition and an out-of-band partition. The out-of-band partition is written in Tcl and controls the in-band media processing modules written in C++. Compute-intensive means that computers perform analysis on multimedia data, and can take actions based on the findings. In our framework, we intend to use visual marker information contained within video frames more extensively. A visual marker might contain any kind of information.

Infopipes[15] proposes an abstraction for building distributed multimedia streaming applications. Components such as sources, sinks, buffers, and filters are defined, and multimedia applications are built by connecting them. In our framework, we explicitly specify the connection among components like Infopipes, but the connections are dynamically changed according to the current situation.

Fault Tolerant CORBA specification[23] allows us t create a replicated object to make a service highly re liable. In the specification, when we adopt the pr

Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)
0-7695-2208-4/04 $20.00 © 2004 IEEE
IEEE COMPUTER SOCIETY

mary/backup scheme, one of the replicated objects actually receive a request. The primary replica is specified in an object reference that is passed to a client. When the object reference becomes invalid, the reference to the primary replica is returned by using the location forward mechanism in the IIOP protocol. The scheme is very similar to our automatic reconfiguration support.

A programmable network[5] allows us to change the functionalities of the network according to the characteristics of each applications. Each entity in a programmable network, like a router, has a programmable interface designed to change the functionalities. In our approach, an application can configure each continuous media component according to the characteristics of the application. The capability is similar to a programmable network.

The LocALE[18] framework provides a simple management interface for controlling the life cycle of CORBA distributed objects. It extends mobility support to the CORBA life cycle management mechanism. Objects can be moved to anywhere in a location domain by the explicit request from a client. On the other hand, our framework provides implicit stream reconfiguration by specifying reconfiguration policy.

3 Design Issues

3.1 Mixed Reality

Mixed reality[1] is a technology concerned with superimposing computer generated graphics into video images capturing the real-world. Several mixed reality applications have been developed and proved the effectiveness of the technology [3]. For example, a surgeon trainee can use the technique to visualize instructions during an operation[12], or a mechanic can use it as a tool for the maintenance and repair of complex machinery[7]. Also, NaviCam[26] has shown that the technology can be used for building many ubiquitous computing applications to enhance our daily life.

Developing mixed reality applications is not easy because of complex algorithms needed for the analysis of video streams and the generation of graphical images. Middleware like ARToolkit[2] and DWARF[4] have been developed to reduce the effort of programmers, but they do not satisfy the requirements for building mixed reality applications for ubiquitous computing, as described in the next section.

3.2 Requirements for Mixed Reality for Ubiquitous Computing

Developing mixed reality applications for ubiquitous computing, the programmer is faced with complexities inherent to ubiquitous computing environments. Existing mixed reality toolkits such as the ARToolkit[2] are not designed for such environments, and do not address these complexities. We found that the following two requirements must be satisfied for building mixed reality applications in ubiquitous computing environments.

High-Level abstraction to hide heterogeneity: Ubiquitous computing environments consist of various types of computers and networks. Networks may contain a mix of resource-constrained and specialized computers. Some computers may not be appropriate for processing heavy computation like video analysis. For example, cellular phones and PDAs are not appropriate for heavy computations, but they might want to utilize mixed reality features. Also, in ubiquitous computing environments, we need to use various types of devices. For example, continuous media applications for ubiquitous computing should take into account various types of cameras, displays, microphones, and speakers. Therefore, application programmers may develop a different application program for each platform and device. A middleware providing high-level abstraction to hide such differences from application programmers is necessary[19, 21] in order to reduce the development costs.

Automatic reconfiguration to cope with an environmental changes: In ubiquitous computing environments, there will be many cameras and displays everywhere, we believe that a middleware infrastructure should provide a mechanism to support dynamic configuration to change machines executing components according to the current situation. Mixed reality applications in such environments should be able to select the most suitable device according to our current situation. For example, a user may want to see a video stream captured by a camera nearest to him on his cellular phone's display. However, implementing automatic reconfiguration in an application directly is very difficult. An application programmer does not want to be concerned with such complexities and therefore we believe that it is desirable to handle automatic reconfiguration in our framework.

4 Middleware supporting Mixed Reality

In this section, we describe the design and implementation of MiRAGe (Mixed Reality Area Generator), the middleware we have developed to support mixed reality for ubiquitous computing.

4.1 Overview of Architecture

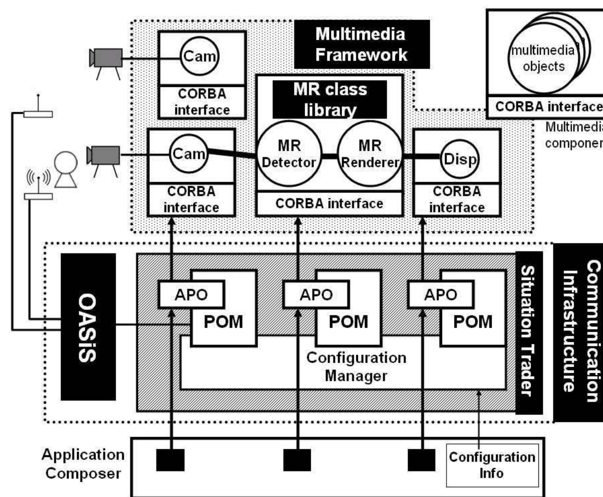

Figure 1: Overview of MiRAGe Architecture

MiRAGe consists of the *multimedia framework*, the *communication infrastructure* and the *application composer*, as shown in Figure 1. The multimedia framework, described in Section 4.2, is a CORBA-based component framework for processing continuous media streams. The framework defines CORBA interfaces to configure multimedia components and connections among the components.

Multimedia components supporting mixed reality can be created from the MR class library. The library contains several classes that are useful to build mixed reality applications. By composing several instances c the classes, mixed reality multimedia components can b constructed without taking into account various comple

[1] Some researchers use the term *augmented reality* rather then mixed reality.

Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems, Networking and Services (MobiQuitous'04)
0-7695-2208-4/04 $20.00 © 2004 IEEE
IEEE COMPUTER SOCIETY

algorithms realizing mixed reality. The MR class library is described in Section 4.4.

The communication infrastructure based on CORBA, described in Section 4.3, consists of the *situation trader* and *OASiS*. The situation trader is a CORBA service that supports automatic reconfiguration, and is collocated with an application program. Its role is to manage the configuration of connections among multimedia components when the current situation is changed. OASiS is a context information database that gathers context information such as location information about objects from sensors. Also, in our framework, OASiS behaves like as a Naming and Trading service to store objects references. The situation trader communicates with OASiS to detect changes in the current situation.

Finally, the application composer, written by an application programmer, coordinates an entire application. A programmer needs to create several multimedia components and connect these components. Also, he specifies a policy on how to reconfigure these components to reflect situation changes. By using our framework, the programmer does not need to be concerned with detailed algorithms for processing media streams because these algorithms can be encapsulated in existing reusable multimedia components. Also, distribution is hidden by our CORBA-based communication infrastructure, and automatic reconfiguration is hidden by the situation trader service. Therefore, developing mixed reality applications becomes dramatically easy by using our framework.

MiRAGe satisfies the requirements described in the previous section in the following way.

High-Level Abstraction: MiRAGe provides a multimedia framework for constructing mixed reality components in an easy way. Complex programs like detecting visual markers and drawing 3D objects are encapsulated in respective multimedia components. Also, detailed characteristics about respective devices are encapslated in components that offer common interface. All components offer an identical CORBA interface for standardized inter-component access. In our framework, a complex distributed and automatically reconfigurable mixed reality application can be developed by writing the application composer program that composes reusable multimedia components.

System-Level Automatic Reconfiguration: In the MiRAGe framework, the communication infrastructure is designed as a CORBA compliant system that supports automatic reconfiguration. The infrastructure supports user mobility by automatically updating object references and reconfiguring media streams. Also, the infrastructure allows us to select the most suitable component to process media streams automatically and to reconnect the component, according to the characteristics of each computer platform and the situation of a user by specifying policies. However, an application program needs not to take into account how an application changes the configuration due to the current situation's changes by using our middleware infrastructure.

4.2 Multimedia Framework

The main building blocks in our multimedia framework are software entities that internally and externally stream multimedia data in order to accomplish a certain task. We call them *multimedia components*. In this section, we describe the components in more detail and provide programs to illustrate how a developer can configure the multimedia components.

4.2.1 Multimedia Components

A multimedia component consists of a CORBA interface and one or more *multimedia objects*. For example, Figure 1 shows three connected components: One component that contains a camera source object for capturing video images, one component that contains the `MRDetector` and `MRRenderer` filter objects for implementing mixed reality functionality as described in Section 4.4, and one component that contains a display sink object for showing the mixed reality video images.

In a typical component configuration, video or audio data are transmitted between multimedia objects, possibly contained by different multimedia components, running on remote machines. Through the CORBA verb—MConnIface— interface, as described in the next subsection, connections can be created in order to control the streaming direction of data items between multimedia objects. Multimedia components register themselves at the CORBA Naming Service under a user-specified name.

4.2.2 CORBA Interface

A component can be remotely accessed through one of three CORBA interfaces: `MCompIface`, `MConnIface` and `MServIface`.

The `MCompIface` interface is added to the component to provide a single object reference through which references can be obtained to other CORBA interfaces. The benefits of adding such an interface is to give clients access to all inter-component functionality through a single reference. In addition, the `MCompIface` interface provides functions to query individual objects and the component as a whole. The `MCompIface` interface is identical to all components.

The `MConnIface` interface provides methods to establish connections between objects, possibly contained by different multimedia components, running on remote sites. More specific, the interface provides functions to create sockets for inter-component streaming, updating the streaming information managed by individual multimedia objects, and to start and stop streams. The `MConnIface` interface is also identical to all components.

The `MServIface` interface provides methods for controlling specific multimedia objects within a multimedia component. Clients may find it useful to query and/or change the state of a multimedia object. For example, a client may want to query a display object for the resolutions it supports and may want to change the resolution to its needs. The `MServIface` interface varies from component to component, depending on the internal multimedia objects it contains.

The interfaces are part of the module IFACE and are written in CORBA IDL. Here follows a snapshot of the module[2] :

```
module IFACE
{
   interface MConnIface
   {
      ObjectId createServerSocket(out SocketInfo info)
      ObjectId createClientSocket(in SocketInfo info)

      void addStreamInfo(in ObjectId nTargetId, in StreamInfo info)

      void startStream(in ObjectId nTargetId, in StreamId nStreamId)
      void stopStream(in ObjectId nTargetId, in StreamId nStreamId)
   };

   interface MCompIface
   {
      MConnIface getConnIface();
      MServIface getServIface();

      boolean isInput(in ObjectId nTargetId)
      boolean isOutput(in ObjectId nTargetId)

      DataType getDataType(in ObjectId nTargetId)
   };
```

[2] The Services interface is not included since it varies for different component configurations. Also, detailed definitions about data types and exceptions are omitted.

};

4.2.3 Multimedia Objects

In our approach, the central focus is the stream of data from data producers to data consumers through zero or more data manipulators similar to VuSystem[16]. Data producers typically are interfaces to video or audio capture hardware or media storage hardware. In our framework we call them *sources*. Data manipulators perform operations on the media data that runs through them. Data manipulators get their data from sources or other data manipulators and stream the modified data to a consumer or another manipulator. In our framework we call them *filters*. Data consumers are multimedia objects that eventually process the data. Data consumers typically interface to media playback devices or to media storage devices. In our framework we call them *sinks*.

More concrete, our framework provides the abstract classes `MSource`, `MFilter` and `MSink`[3] . Developers extend the classes and override the appropriate hook-methods to implement functionality. Multimedia objects need only to be developed once and can be reused in any component.

The multimedia framework defines two specialized classes of multimedia objects for handling inter-component data streaming, namely `MClientSocket` and `MServerSocket`. Socket objects can be created and connected through the appropriate function calls defined in the CORBA `MConnIface` interface.

4.2.4 Streams

A typical mixed reality component might contain a filter object that adds digital images to video frames at specified positions within the frames. Different client components may want to use the service at the same time by sending video frames to the component and afterwards receiving it for playback. This implies that different data items streamed through filter objects within multimedia components might have different destinations. Solely setting up direct connections between objects does not satisfy the above described scenario. If each client would be connected to the filter object as a destination, how does the filter object know which data is to be send to which destination?

To solve the above issue we do not use direct connections between multimedia objects. Rather, we assign a unique *stream identifier* to each stream and use *stream tables* managed by output objects to hold stream direction information. Source objects add a stream identifier to each data item they produce, identifying the stream the data is part of.

The stream table managed in the outport of each output object store tuples of type `[StreamId, ObjectId]`. The stream identifier sent with each data item is used for finding the target multimedia object. If found, the data is send to the inport of the target object and put in the appropriate buffer, also identified by its stream identifier.

Our framework defines a `MStream` class that acts as a facade for the primitive low-level CORBA `MConnIface` interface functions. It provides easy means for the developer to set up a distributed stream between multimedia objects. The interface defines methods for setting the source and sink object of a stream and adding one or more filters. In addition, methods are added to start and stop a media stream.

In order to identify a multimedia object within a component, the framework assigns a unique object identifier to each object after it is added to the component. Universally, we use a tuple of type `[MCompIface, ObjectId]` to denote one specific object. Such *universal identifiers* are used by `MStream` objects to compose a stream.

4.2.5 Component Configuration

In our framework, we use a component abstraction that hides much of the details that deal with CORBA and streaming. By extending the abstraction, a developer can configure a component. More specific, a developer specializes the `MComponent` class provided by the framework. In its constructor it typically creates multimedia objects, possibly creates a stream and finally adds the objects to the container component. a program for the example component in Figure 2 might look something like this:

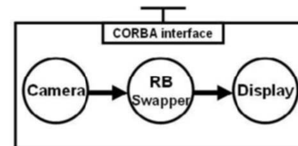

Figure 2: Example Component

```
MyComponent::MyComponent():
   MComponent()
{
   m_pCamera = new Camera;
   m_pSwapper = new RBSwapper;
   m_pDisplay = new Display;

   addObject(m_pCamera);
   addObject(m_pSwapper);
   addObject(m_pDisplay);
}

MCompIface pComponent = MNaming::resolve(''Some_Name'');

MStream stream(MStream::NORMAL);
stream.setSource(pComponent, 1);
stream.addFilter(pComponent, 2);
stream.setSink(pComponent, 3);
stream.start();
```

The above illustrative code retrieves a CORBA object reference from the Naming Service registered under the name `Some_Name` and assumes such a reference exists. Next, a stream with normal priority is set up between the participating multimedia objects[4] . After the stream is started, data is streamed from the camera to the display object through the red-blue swapper.

4.3 Communication Infrastructure

Our CORBA-based communication infrastructure consists of two subsystems, namely the *situation trader* and *OASiS*, as described in Section 3.1. Complex issues about automatic reconfiguration are handled by the situation trader and therefor hidden from the application programmer. A *configuration manager*, owned by the situation trader, manages stream reconfiguration by updating connections between multimedia objects. The situation trader is linked into the application program.

In our framework, a proxy object in an application composer refers to an *Adaptive Pseudo Object or APO*, managed by the situation trader. Each APO is managed by exactly one *Pseudo Object Manager or POM* that is responsible for the replacement of object references by receiving a notification message from OASiS upon situation change.

4.3.1 Automatic Invocation Forwarding

In our system, an application programmer uses a POM to access multimedia components. Each POM manages one APO, a CORBA object that has the same interface as its target multimedia object. The APO forward

[3] The M preceding the class names indicate that they are part of the framework and stands for multimedia.

[4] In the example we assume the object identifiers are known in advance.

client invocations to the most appropriate target component. The application programmer can specify a reconfiguration policy with a POM to control the reconfiguration strategy. Upon startup, a POM retrieves an initial IOR from OASiS. OASiS finds the most appropriate object reference from its context information database according to the registered reconfiguration policy and updates the reference upon situation change.

Figure 3 shows how dynamic invocation forwarding works, and the following describes the depicted sequence in more detail.

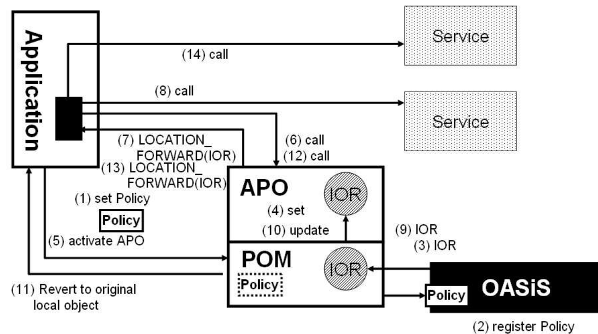

Figure 3: Automatic Invocation Forwarding

An application registers a reconfiguration policy with a POM (1). The POM passes the policy to OASiS (2). OASiS returns an initial IOR that is appropriate for the current situation (3). The POM passes the IOR to its APO (4). The application requests the activation of the APO (5). The application invokes a method on the APO (6). The APO returns a LOCATION_FORWARD message containing the IOR in the POM (7). The application resends the previously issued request by using the enclosed IOR (8). When the current situation changes, OASiS notifies a new IOR that is appropriate for the new situation (9). The POM updates the received IOR in the APO (10). Then, the POM reverts the current object reference, and the object reference of a new target object is retrieved by using a LOCATION_FORWARD message again (11). Thus, a client invocation is forwarded transparently according to the new situation(12, 13, 14).

4.3.2 Situation Trader Service:

Figure 4 shows the relation between the situation trader and the continuous media framework. As described in the overview, the situation trader is a CORBA service that manages the reconfiguration of multimedia components. This subsection presents in detail how the situation trader works.

The following sample C++ code illustrates how an application program might look like:

```
CORBA::Object_var obj =
  orb->resolve_initial_reference("SituationTraderService"); //(1)
STrader::SituationTraderFactory_var factory =
  STrader::SituationTraderFactory_narrow(obj);

STrader::POManager_var camera_pom =
  factory->createPOManager("IFACE::MCompIface:1.0"); //......(2)
STrader::POManager_var display_pom =
  factory->createPOManager("IFACE::MCompIface:1.0");
STrader::ConfigurationManager_var com =
  factory->createConfigurationManager();

STrader::ReconfigurationPolicy camera_policy;
STrader::ReconfigurationPolicy display_policy;

camera_policy.locationScope = "Distributed Computing Laboratory"
camera_policy.locationTarget = "Eiji TOKUNAGA"
camera_policy.locationContext = "nearest"
camera_policy.serviceType = "Camera" ......(3)

display_policy.locationScope = "Distributed Computing Laboratory"
display_policy.locationTarget = "Andrej van der Zee"
display_policy.locationContext = "nearest"
```

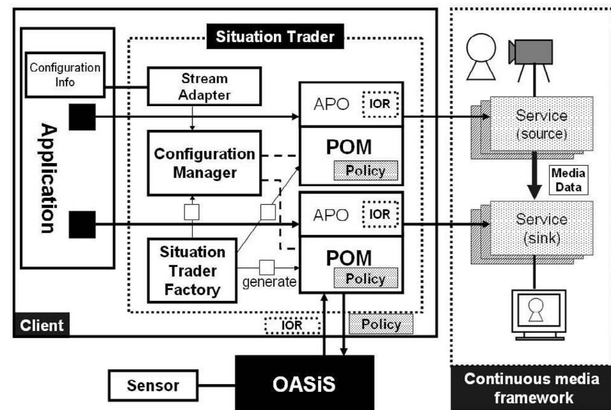

Figure 4: Situation Trader Service

```
display_policy.serviceType = "Display"

camera_pom->setPolicy(camera_policy); //......(4)
display_pom->setPolicy(display_policy);

IFACE::MCompIface_ptr camera_apo =
  camera_pom->activateAPObject();
IFACE::MCompIface_ptr display_apo =
  display_pom->activateAPObject();

MStream stream(MStream::NORMAL);
stream.setSource(camera_apo, 1);
stream.setSink(display_apo, 1);

StreamAdapter_i* adapter_i = new StreamAdapter_i(stream); //......(5)
STrader::ConfigurationAdapter_ptr adapter = adapter_i->_this();

com->setAdapter(adapter); //......(6)
com->addPOManager(camera_pom); //......(7)
com->addPOManager(display_pom);

// Start streaming.
stream.start();
```

As shown in Figure 4, the situation trader consists of a *situation trader factory*, a *configuration manager* and several POMs. The object reference to the situation trader is retrieved by invoking the `resolve_initial_reference` method provided by CORBA (line 1).

The situation trader factory is used for creating POMs and the configuration manager (line 2). The method `createPOManager` expects as parameter the ID of the target object that specifies the object's type, and returns a reference to the POM that manages the APO.

A reconfiguration policy needs to be set for each POM (line 4). The policy is passed to OASiS through the POM, and OASiS selects the most appropriate target object according to the policy.

In the current design, a reconfiguration policy has three location parameters, *locationScope*, *locationTarget* and *locationContext* (line 3). LocationScope denotes the scope for selecting a suitable multimedia component. When a POM passes the policy to OASiS, OASiS searches a target multimedia component in the specified scope. LocationTarget specifies a physical object used to represent the current situation. LocationContext specifies the relation between a target multimedia component and the physical object specified by locationTarget. LocationTarget might be a person's name or a device's name. Currently, locationContext can specify "nearest" and "exact". "Nearest" means that the nearest multimedia component to a physical object specified by locationTarget should be selected. For example, if locationContext is "nearest" and locationTarget is "Eiji TOKUNAGA", this pair means "nearest to Eiji TOKUNAGA". "Exact" means that a multimedia component that resides with a physical object specified by locationTarget should be selected. We are considering to define more policies in the future version of our framework.

Several POMs can be added to the configuration manager. The configuration manager retrieves the APO

IEEE COMPUTER SOCIETY

from its registered POMs and controls automatic reconfiguration of the APOs. A stream adapter needs to be set for the configuration manager for automatic stream reconfiguration (line 6). When one of POMs is updated by OASiS, the stream adapter reconfigures connections between multimedia components in order to reflect situation change (line 5 to 7). Stream reconfiguration is explained in more detail in the next subsection.

4.3.3 Stream Reconfiguration

A stream adapter controls the `MStream` object described in Section 4.2. Upon situation change, a callback handler in the stream adapter is invoked in order to reconfigure affected streams through its `MStream` object.

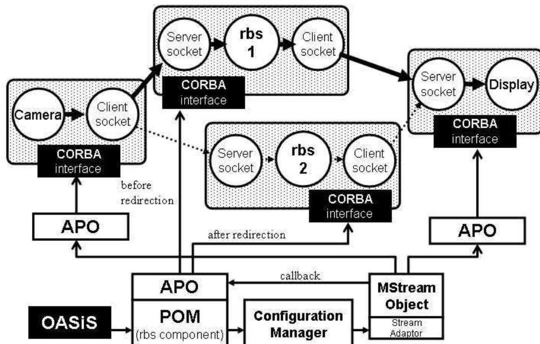

Figure 5: Stream Reconfiguration

Figure 5 depicts how connections among multimedia objects are changed in order to reconfigure an existing stream. In the figure, the camera object streams media data to the display object through the red-blue swapper `rbs1`. When current situation changes and the red-blue swapper `rbs2` becomes the most appropriate object, the callback handler for the stream adapter is invoked passing the old and the new POM, that is `rbs1` and `rbs2`. The callback handler updates the `MStream` object and restarts the stream.

More concrete, the `MStream` object controlled by the stream adapter stops the old stream by removing its stream identifier from its source object. Next, the old reference is replaced by the new one. Finally, the newly configured `MStream` object is restarted. Internally, restarting is done by setting up the appropriate TCP connections between remote components, updating the stream information of the participating objects and adding a new unique stream identifier to the source object.

4.4 Mixed Reality Components

The *MR class library*, as shown in Figure 6, is part of the MiRAGe framework. The library defines *multimedia mixed reality objects* for detecting visual markers in video frames and superimposing graphical images on visual markers in video frames. These mixed realtity multimedia objects are for a large part implemented using the ARToolkit. Application programmer can build mixed reality applications by configuring multimedia components with the mixed reality objects and stream data between them. In addition, the library defines data classes for the video frames that are streamed through the MR objects.

`MRFilter` is a subclass of `MFilter` and is used as a base class for all mixed reality classes. The class `MVideoData` encapsulates raw video data. The `MRVideoData` class is a specialization `MVideoData` and contains a `MRMarkerInfo` object for storing information about visual markers in its video frame. Since different types of markers will be available in our framework,

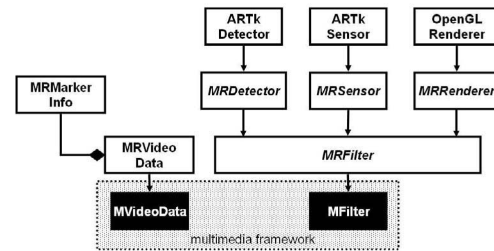

Figure 6: Mixed Reality Class Library

the format of marker information must be defined in a uniform way.

The class `MRDetector` is a mixed realty class and inherits from `MRFilter`. The class expects a `MVideoData` object as input and detects video markers in the `MVideoData` object. The class creates a `MRVideoData` object and adds information about detected markers in the video frame. The `MRVideoData` object is send as output. The class `ARTkDetector` is a subclass of `MRDetector` that implements the marker detection algorithm using the ARToolkit.

The `MRRenderer` class is another mixed reality class derived from `MRFilter`. The class expects an `MRVideoData` as input and superimposes graphical images at positions specified in the `MRMarkerInfo` object. The superimposed image is send as output. The `OpenGLRenderer` is a specialization of `MRRenderer` and superimposes graphical images generated by OpenGL.

The `MRSensor` class is a specialization of `MFilter` and sends the current marker information to OASiS for the detection of the location of a physical object. `ARTkSensor` inherits from `MRSensor` and uses the ARToolkit for its implementation.

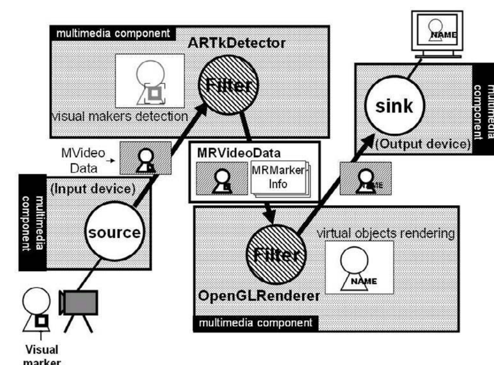

Figure 7: An Example MR Application

Figure 7 illustrates how mixed reality components can be configured and connected in an application. In the example, a *visual marker* attached to a person is captured and superimposed by information about this persons profile before display.

In detail, the camera object sends `MVideoData` objects, representing the captured video frames, to the visual marker detector object. The detector object adds information about visual tags to a `MRVideodata` object and sends it to the superimposer object. The superimposer object substitutes digital images for visual markers and sends the superimposed video frames to the display object.

If a powerful computer is available, the two filter components for detection and superimposing can be merged into one component. Our framework can create multimedia components that are suitable for respective platforms.

5 Sample Scenarios

This section describes two scenarios showing the effectiveness of MiRAGe. In the first scenario, we describe a *follow-me* application that dynamically changes camera and display devices according to user location. In the second scenario, we describe how *mobile mixed reality* can be used on less powerful devices such as PDAs and cellular phones.

5.1 A Follow-Me Application

In this section, we consider an application that receives a video stream from a camera and displays it on the nearest display to the user. As shown in Figure 8, there are two continuous media components. The first one is a camera component, and the second one is a display component. The two components are connected by an application composer. However, the actual display component is changed according to user location. An application composer holds a POM managing several display objects and changes the target reference of an APO to a display nearest to the user. A configuration manager reconfigures a stream when the current situation is changed.

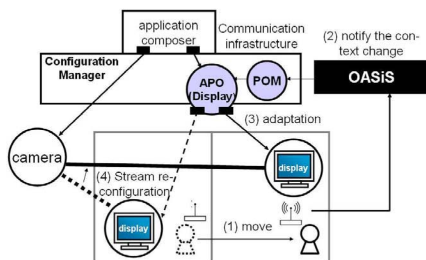

Figure 8: A Follow-me Application

When the user moves, a location sensor detects the movement of the user. As a result, OASiS is notified by the location sensor (1). OASiS notifies an IOR of the nearest display to the POM, then the POM changes the target reference in the APO (2). Therefore, a method invocation is forwarded to the nearest display component (3). In this case, when a callback handler in the configuration manager is invoked, the configuration of the stream is changed (4).

5.2 Mobile Mixed Reality

In a typical mobile mixed reality application, our real-world is augmented with virtual information. For example, a door of a classroom might have a visual tag attached to it. If a PDA or a cellular phone, equipped with a camera and an application program for capturing visual tags, the tags are superimposed by a schedule of today's lecture.

We assume that in the future our environment will deploy many mixed reality servers. In the example, the nearest server stores information about today's lecture schedule and provides a service for detecting visual tags and superimposing them by the information about the schedule, as depicted in Figure 9.

Other mixed reality servers, located on a street, might contain information about what shops or restaurants can be found on the street and until how late they are open.

To build the application, an application composer uses components for capturing video data, detecting visual markers, superimposing information on video frames and displaying them. The application composer contacts a situation trader service to retrieve a reference to a POM managing references to the nearest mixed reality server to a user. When he moves, a location sensor

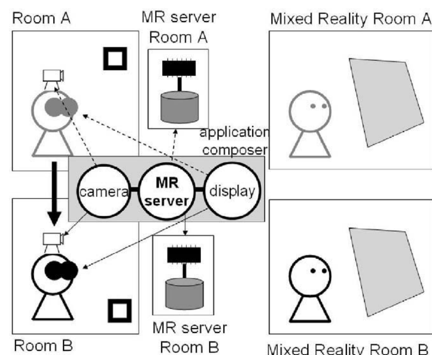

Figure 9: Moblie Mixed Reality

component notifies sensed location information to OASiS, and OASiS notifies the situation trader to replace the current object reference to the reference of the nearest mixed reality server. In this way, the nearest mixed reality server can be selected dynamically according to his location, but the automatic reconfiguration is hidden from an application programmer.

6 Current Status

In our current prototype, we are using C++ and omniORB [17] for our CORBA-based communication infrastructure. OmniORB is open source and very efficient. In our design of continuous media components, respective multimedia objects run in separate threads. Therefore, a fully multi-threaded CORBA compliant ORB is required.

We describe the evaluation to show the effectiveness of our approach in this section. Also, we show an actual implementation of a sample scenario described in the previous section and present some discussions about the current prototype implementation.

6.1 Evaluation of Distributed Mixed Reality

The section presents the evaluation showing the effectiveness of automatic reconfiguation supported by MiRAGe. The result shows the impact to delegate heavy computation to a powerful server.

The picture shown in Figure 10 presents an evaluation environment of distributed mixed reality. The laptop computer on the right side has PentiumIII 866MHz processor and 256MBytes memory. The left one is a high performance server computer which has Pentium4 1.9GHz processor and 512MBytes memory. Our infrastructure is currently running on Linux. These computers are connected by the 100Base-T Ethernet. A user with the laptop computer can watch superimposed video images on its screen.

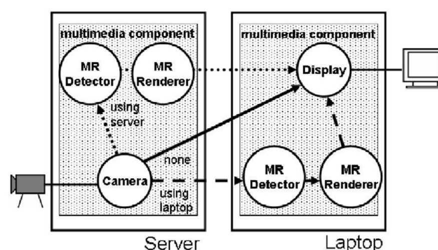

Figure 10: Three Cases of MR Stream

To evaluate the merit of distribution in our component based approach, we compared the performance i

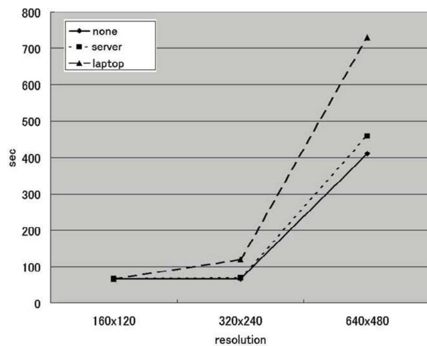

Figure 11: Processing Time for 2000 Frames

three cases shown in Figure 10. The graph in Figure 13 shows the time required to display 2000 superimposed video frames on the laptop's display when a data source generates 30 video frames per second. In the evaluation, "none" means that a captured video image by the server is rendered by the server without mixed reality processing, "server" means that mixed reality processing runs on the server, and "laptop" means that mixed reality processing runs on the laptop. The result shows that the processing time to analyze video images and to superimpose a graphic image on them on the laptop computer is dramatically increased according to the data size. On the other hand, when using a powerful server to execute mixed reality processing, the heavy computation does not affect the performance seriously. Therefore, our approach that delegates heavy computation to a powerful server near a user will improve the performance of mixed reality applications significantly.

In the evaluation, we have adopted a high bandwidth network, but when we use low bandwidth networks, we can add a component to decrease the quality of video streams to reduce the bandwidth before transmitting the video streams to the low bandwidth networks. However, we may use high handwidth networking adopting UWB(Ultra Wide Band) technologies even by small mobile devices in the near future.

6.2 Moblie Mixed Reality Prototype

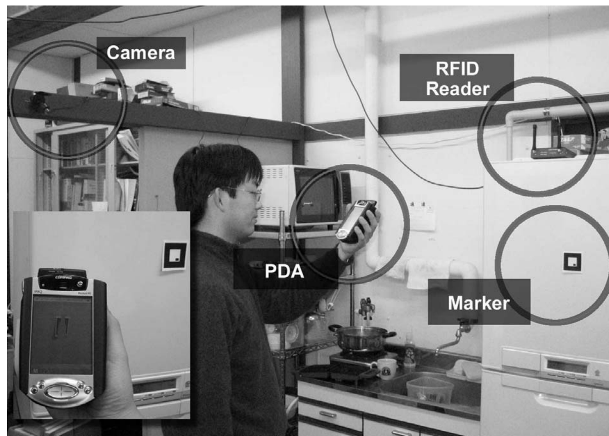

Figure 12: Mobile Mixed Reality Prototype

Figure 12 is a picture showing a prototype of a mobile mixed reality application. The PDA, that the person in the picture has in his hand, is a Compaq iPAQ H3800 with a wireless LAN card and a TOSHIBA 2GB PCCARD hard disk. We attached a RFID tag to this PDA for detecting its location. The refrigerator on the right side is a TOSHIBA IT refrigerator named *Feminity*. The refrigerator is equipped with sensors that let us know how many bottles are inside.

The following a simplified program for the mobile mixed reality prototype.

```
camera_policy.locationScope = "Laboratory"
camera_policy.locationTarget = "Feminity"
camera_policy.locationContext = "nearest"
camera_policy.serviceType = "Camera"

filter_policy.locationScope = "Laboratory"
filter_policy.locationTarget = "Feminity"
filter_policy.locationContext = "nearest"
filter_policy.serviceType = "MRFilter"

display_policy.locationScope = "Laboratory"
display_policy.locationTarget = "Feminity"
display_policy.locationContext = "nearest"
display_policy.serviceType = "Display"

camera_pom->setPolicy(camera_policy);
filter_pom->setPolicy(filter_policy);
display_pom->setPolicy(display_policy);

MStream stream(MStream::NORMAL);
stream.setSource(camera_apo, 1);
stream.addFilter(filter_apo, 1);
stream.addFilter(filter_apo, 2);
stream.setSink(display_apo, 1);

Stream_i* adapter_i = new StreamAdapter_i(stream);
STrader::ConfigurationAdapter_ptr adapter =  adapter_i->_this();

com->setAdapter(adapter);
com->addPOManager(camera_pom);
com->addPOManager(filter_pom);
com->addPOManager(display_pom);

stream.start();
```

In this scenario, when a user comes near the refrigerator, the RFID reader recognizes the RFID tag attached to the PDA. Then, the RFID reader sends the tag information to OASiS. OASiS recognizes the situation's change, and notifies the IOR of a display service running on the PDA to the mobile mixed reality application that shows video frames on the nearest display service to the refrigerator.

In this case, a mixed reality component that detects visual tags and superimposes digital images on video frames is running on a powerful machine not shown in the picture. In the example, the mixed reality component retrieves the number of bottles from the refrigerator and superimposes a graphical image showing this number.

Currently, a cellular phone in Japan has a camera, and the next version of the cellular phone will adopt sophisticated operating systems such as Linux, and provide a wireless LAN support. We believe that our middleware infurastructure can be used on the cellular phone to offer new services.

7 Discussions

We believe that our design is very effective. Especially, the automatic reconfiguration of an application to reflect a situation change seems very promising. However, in our approach, a number of issues still need to be addressed. In this section, we will discuss strengths and weaknesses of our current design.

7.1 Reconfiguration Policy

In our approach, an application programmer needs to specify reconfiguration policies to reflect application behavior to his or her desire. Most infrastructure softwar for ubiquitous computing adopt a different approacl. For example, the Context Toolkit[6] offers a mechanisı

Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)
0-7695-2208-4/04 $20.00 © 2004 IEEE

to deliver events upon situation change, making the development of context-aware applications more difficult. In our approach, we choose to hide as much detail as possible from the programmer in order to reduce the development costs of ubiquitous computing applications.

Some policies are very difficult to implement, however. For example, assume a location policy that always chooses a nearest service to user location. The presence of a wall between a user and a server inflicts complications regarding to implementation. Clearly, the nearest server might not be the most suitable one. We need to investigate a way to specify a policy that does not depend on sensor technologies to monitor situation information.

We are currently in the process of extending stream reconfiguration abstraction. We will adapt TOAST[8] like hierarchical concept of stream binding for recursive dynamic reconfiguration. In this concept, an application programmer needs to specify reconfiguration policies for stream objects as binding object. For example, she can specify that the continuous media stream must choose a camera device nearest to the user and a filter service nearest to the user. Then, the stream object dynamically creates lower layer binding objects reflecting specific location-aware technologies such as RFID, Wireless LAN, Supersonic and Vision-Based Tracking. In this case, the binding object of the camera device is Vision-Based Tracking and the another one can choose any binding objects. We believe such a hierarchical binding abstraction can provide appropriate high-level abstraction and flexibility.

7.2 ID Recognition-Based Interaction

Using our framework, we have implemented a vision-based direct interaction application like u-Photo[14] . In this application, using camera-attached mobile devices, we can get the graphical user interface of controlling particular embedded devices, such as remote control GUI of VCR, through actions of taking images of visual markers on them. It is easy to take images by the mobile devices and send them to the remote visual marker recognition filter due to our framework abstraction. In such a Mixed Reality application domain, however, applications have to recognize adding, removing and updating of visual tags in the capturing images and handle events relevant to these actions.

The implementation of these functions needs some programmers' effort, because our framework does not provide any abstractions for firing and handling visual tag recognition events, although we can handle visual tag recognition processes in our Filter Object. In future study, we will incorporate a high level event model for ID recognition and utilization like Papier-Mache[13] into our framework. The event flow in that model should be reconfigurate as same style as stream reconfiguration.

7.3 Media Recognition-Based Interaction

Now we have built a simple gesture recognition-based application on our framework. That recognizes a user's hand movement and followly moves the active window on the user's nearest X-Window System. In this case, the application needs to accumulate several video frames and analyze differences in them. It is easy to select a camera object to capture the user's hand movement since it is dynamically adapted by an automatic reconfiguration policy specified as "nearest to user".

Other development costs rise, however, on the recognition process. Because the gesture recognition process on the Filter Object must initialize buffers to hold video frames and manage them, while our framework does not provide any programming model handling multiple video frames. There are the same problems in most media recognition applications, such as speech recognition. We need to take account of an appropriate programming model handling multiple subsequences.

7.4 Component State Management

In our framework, we assume that continuous media components do not have states. Consequently, if multimedia components are reconfigurated as a result of a situation change, restoring state information of new target components is not necessary. However, we found that components controlling devices might hold some state information to configure device parameters such as the resolution of images. Currently, in our approach, the application composer restores such parameters after receiving a change-of-situation notification. In this case, we must describe application specific restoring process from scratch when building new multimedia components and application composers. The approach increases the development cost. The state restoring processes should be managed automatically.

Using the CORBA Persistent Service[22] and moving components states according to stream reconfiguration may be a solution if there are many stable ORBs providing it, but there are not so many ORBs providing stable and fully interoperable PSS. Therefore, we plan to utilize Ice(Internet Communication Engine)[9] instead of CORBA as our next communication infrastructure base. Ice provides a next-generation language independent object-oriented distributed environment and the build-in stable persistent state service.

7.5 System Independent Framework

Our framework provides the situation trader service to reconfigure connections among multimedia components automatically. Our approach does not require to modify CORBA runtime. Thus, it is easy to port our framework in different CORBA systems and other object-oriented middleware such as Ice. The next design extensions described above will be ORB independent as well.

On the other hand, the current implementation of our context information database OASiS is rather system dependent because it combines ORB specific services such as CORBA Naming Service and CORBA Trading Service. And it dose not provide system independent subscription and notification framework. Therefore, we need to redesign OASiS as a combination of publish/subscribe system like EventHeap[11] and context information database including naming service in system independent way.

We believe XML-based messaging protocol like XMPP[10] or Macromedia Flash XML Socket is appropriate for both of publish/subscribe transaction and registering naming service. These XML-based messaging protocols have highly extensibility and system independence by XML's nature. XML-based messaging context information database will be discussed in the future study.

8 Conclusion

In this paper, we have described our middleware framework to support mixed reality for ubiquitous computing. We have described design and implementation of our system, and presented some experiences with our current prototype system. Our experiences show that our system is very useful to develop several mixed reality applications for ubiquitous computing.

In the future, we like to continue to improve our middleware framework, and to develop attractive mixed reality applications such as game, navigation, and enhanced communication applications. Currently, our system is running on Linux, and we like to exploit real-tim

IEEE COMPUTER SOCIETY

capabilities provided by Linux to process video streams in a timely fashion. Also, we are interested to take into account to use a device proposed in [24] since the device can augment the real world without a display by projecting computer generated graphics on real objects directly.

References

[1] G.D. Abowd, E.D. Mynatt, “Charting Past, Present, and Future Research in Ubiquitous Computing”, ACM Transaction on Computer-Human Interaction, 2000.

[2] ARToolkit, http://www.hitl.washington.edu/people/grof/SharedSpace/Download/ARToolKitPC.htm.

[3] R.T. Azuma, “A Survey of Augmented Reality”, Presence: Teleoperators and Virtual Environments Vol.6, No.4, 1997.

[4] Martin Bauer, Bernd Bruegge, et al.: *Design of a Component-Based Augmented Reality Framework*, The Second IEEE and ACM International Symposium on Augmented Reality, 2001.

[5] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, Daniel Villela, “A Survey of Programmable Networks”, ACM SIGCOMM Computer Communications Review, Vol.29, No.2, 1999.

[6] A.K.Dey, G.D.Abowd, D.Salber, “A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications”, Human-Computer Interaction, Vol.16, No.2-4, 2001.

[7] Steven Feiner, Blair MacIntyre, and Doree seligmann. “Knowledge-based Augmented Reality”, Communications of the ACM 36, 7 (July 1993) , 52-62

[8] Fitzpatrick, T., Gallop, J., Blair, G.S., Cooper, C., Coulson, G., Duce, D., Johnson, I., “Design and Implementation of TOAST: An Adaptive Multimedia Middleware Platform”, Proceedings of IDMS'01, Lancaster, UK, September 2001.

[9] Michi Henning, “A New Approach to Object-Oriented Middleware,” IEEE Internet Computing, January-February 2004, pp 66-75

[10] IETF Internet-Draft “Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence,” http://www.ietf.org/internet-drafts/draft-ietf-xmpp-im-22.txt

[11] Brad Johanson and Armando Fox, “Extending tuplespaces for coordination in interactive workspaces”, Journal of Systems and Software, v.69 n.3, p.243-266, 15 January 2004

[12] Anantha R. Kancherla, Jannick P. Rolland, Donna L. Wright, and Grigore Burdea. “A Novel Virtual Reality Tool for Teaching Dynamic 3D Anatomy”, Proceedings of Computer Vision, Virtual Reality, and Robotics in Medcine '95 (CVRMed '95) April 1995.

[13] Scott Klemmer, “Papier-Mache: Toolkit support for tangible interaction,” in proceedings of The 16th Annual ACM Symposium on User Interface Software and Technology: UIST 2003 Doctoral Consortium.

[14] N. Kohtake, T. Iwamoto, G. Suzuki , S. Aoki, D. Maruyama, T. Kouda, K. Takashio, H. Tokuda, “u-Photo: A Snapshot-based Interaction Technique for Ubiquitous Embedded Information,” Second International Conference on Pervasive Computing (PERVASIVE2004), Advances in Pervasive Computing, 2004

[15] R.Koster, A.P. Black, J.Huang, J.Walpole, and C.Pu, “Thread Transparency in Information Flow Middleware”, In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, 2001.

[16] Christopher J. Lindblad, David L. Tennenhouse: *The VuSystem: A Programming System for Compute-Intensive Multimedia*, In Proceedings of ACM International Conference on Multimedia 1994.

[17] S Lo, S Pope, “The Implementation of a High Performance ORB over Multiple Network Transports”, In Proceedings of Middleware 98, 1998.

[18] Diego Lopez de Ipina and Sai-Lai Lo, “LocALE: a Location-Aware Lifecycle Environment for Ubiquitous Computing”, In Proceedings of the 15th IEEE International Conference on Information Networking (ICOIN-15), 2001.

[19] T.Nakajima, “System Software for Audio and Visual Networked Home Appliances on Commodity Operating Systems”, In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, 2001.

[20] T.Nakajima, H.Ishikawa, E.Tokunaga, F. Stajano, “Technology Challenges for Building Internet-Scale Ubiquitous Computing”, In Proceedings of the Seventh IEEE International Workshop on Object-oriented Real-time Dependable Systems, 2002.

[21] T.Nakajima, “Experiences with Building Middleware for Audio and Visual Netwoked Home Appliances on Commodity Software”, ACM Multimedia 2002.

[22] OMG, “CORBAServices Specification,” http://www.omg.org/technology/documents/corbaservices_spec_catalog.htm

[23] OMG, “Final Adopted Specification for Fault Tolerant CORBA”, OMG Technical Committee Document ptc/00-04-04, Object Management Group (March 2000).

[24] C.Pinhanez, “The Everywhere Display Projector: A Device to Create Ubiquitous Graphical Interfaces”, In Proceedings of Ubicomp'01, 2001.

[25] K.Raatikainen, H.B.Christensen, T.Nakajima, “Applications Requirements for Middleware for Mobile and Pervasive Systems”, Mobile Computing and Communications Review, Octorber, 2002.

[26] Jun Rekimoto, “Augmented Interaction: Interacting with the real world through a computer” , HCI International, 1995.

[27] M. Weiser, “The Computer for the 21st Century”, Scientific American, Vol. 265, No.3, 1991.

IEEE COMPUTER SOCIETY