ongoing technological advances in distributed systems have made
services much more software based, thus moving the focus of ser-
vice development from telephonic switch environments to more
traditional computer environments. At the same time, broadband networks
have evolved to support more sophisticated applications that include mul-
timedia data exchange, such as computer-supported collaborative work
(CSCW). The ability to rapidly create and deploy such applications depends
heavily on a software architecture’s support for the dynamic cooperation of
stand-alone programs acting as independent components. Beyond this, if
the applications are to run on an open platform, the first challenge is to
extend the platform to address, transparently for programmers, multimedia
and networking issues. We are especially interested in Web-based platforms
because the Web is becoming the common denominator for information
access and exchange, both on the Internet and within the enterprise.

There are currently several tools to help design multimedia applications
over the Web, but most address only the integration of animated images
into Web pages. Sun has proposed the Java Media Framework API as a col-
lection of classes that enable the synchronization, display, and capture of
time-based data within Java applications and applets. JMF offers a multi-
platform framework for multimedia programming, although it lacks hard-
ware device control and predefined classes for multiparty collaboration.

This article summarizes the multimedia and networking issues
addressed in the MultiTel framework for multimedia services (a detailed
discussion will be published later this year by John Wiley & Sons). Multi-
Tel supports a distributed, compositional platform that manages multi-

The MultiTel compositional
framework for developing
collaborative applications
separates communication from
data-processing components
and also designates a
Web-based distributed
platform for intercomponent
communication.
media and networking resources used in the design of complete multimedia collaborative services. An Internet-based infrastructure, MultiTel provides a mechanism that abstracts the details of communication and collaboration between components and their deployment sites, allowing the runtime composition of different multimedia products, chats, or GUIs. This mechanism supports various network protocols, component locations, and so forth.

MULTITEL PLATFORM ARCHITECTURE

Component platforms like DCOM and JavaBeans allow the construction of independent software components, which can further be assembled via application-level frameworks. Not only does the MultiTel framework offer a component platform, but it also constitutes an application framework for the composition of Java components. Components, implemented as Java objects, will run on a distributed platform that uses Java/RMI and Web services. Initially, we tried to develop an open service model that reflected, at the design level, the collaboration and synchronization needed between service components. We subsequently settled on compositional models that place coordination and data processing in different components.

Perhaps the main benefit of component-oriented technology for multimedia programming is the dynamic composition of multimedia devices as components developed by different manufacturers. In addition, it gives components standard interfaces that can be connected easily to make a new multimedia application.

Layered Structure

The MultiTel platform is structured in two different layers: application components that are dynamically composed at runtime and a middleware platform kernel that provides common services for controlling multimedia data delivery.

As shown in Figure 1, the application component has two parts:

- Application Level. Multimedia services are executed by the dynamic composition of service, multimedia, and network components, which are retrieved from the network. Service components model user interactions—that is, the service logic. They are application dependent and therefore not truly part of the middleware platform. Multimedia and network components, on the other hand, primarily model multimedia devices and multicast connections. Because these components represent the user's real resources, the local kernel must bind them.

- User-Service Part. The USP object supports the mechanisms for interactions between service, multimedia, and network components that enable the basic communication process. Users join a multimedia application by executing a USP applet from a browser. They interact with the application through a USP component that represents the local service access point; this component encapsulates the application architecture, which in turn drives the dynamic composition of the distributed components. The USP object can also modify its internal structures in order to add external components, such as vendor plug-and-play components.

The kernel of the middleware platform includes tools for sharing and running applications over the Web. One component, the Resource Manager, acts as a mediator between application components and users' local resources. Another, the Application Directory, is a common service component that shows and manages the list of registered applications.

Compositional Model

The MultiTel framework includes the implementation of a compositional model in Java. This model provides the architectural basis of the derived multimedia applications. In it, the first-level entities are...
components and connectors. Components model real entities, such as multimedia devices and the connectors that implement communication protocols between two or more components. Because this model's main goal is to integrate heterogeneous and distributed components into the same multimedia application, components report status changes by propagating an event to the execution environment.

Components are passive entities characterized by the complete ignorance of how propagated events influence the execution of a particular multimedia service. This approach provides collaboration transparency through event-selective broadcasting, in which a conference connector—by catching events—can synchronize independent instances of components according to the application coordination protocol. In fact, the platform discards those events not considered relevant to the service, allowing the plugging of legacy code into applications.

Alternatively, connectors are abstractions of coordination patterns that implement coordination protocols. By handling the events propagated by external components, a connector controls the execution of any component. Connectors encapsulate the state transition diagram (STD) of a protocol that is triggered by a component's events. The implementation of the connector's protocols follows the State design pattern. In Figure 2, the readyToRcv event propagated by a MultimediaDevice component is caught by the ContCom connector, which responds by sending the write message to the component identified inside the application architecture by the name “device”. Sending events and messages in independent threads allows the concurrent manipulation of components (like GUIs) and connectors.

Not only does the USP object dynamically compose components and connectors, but it also encapsulates the application architecture (how the components have to be plugged into the connectors) and the current application context (the components and connectors instantiated for every user). Figure 2 shows how the USP object decides the appropriate connector (or connectors) to handle the readyToRcv event. The USP also implements methods for message and event broadcasting, as well as for message passing from connectors to components. Since connectors need only know the role of the target component in the service architecture, they could send messages to separately downloaded components that fulfill the required role (“device” in Figure 2). Component and connector roles are defined at the architectural level and stored as part of the USP data.

Finally, added to the middleware platform are component migration capabilities for handling user or application mobility. The Java API provides object serialization classes that facilitate the implementation of components and connector migration, which later restart from a previously stored state. During code migration, pending events are queued by the source USP and then sent to the target one.

**APPLICATION COMPONENTS**

The MultiTel software architecture offers reusable software components and hides the details of underlying technologies and the complexity of collaboration patterns from the service designer. It has three main subsystems: service components, multimedia components, and network components.

**Service Architecture**

The service architecture defines the service logic, that is, the user interactions specialized for specific services. There are three principal actors involved in the MultiTel service provisioning process:

- **Organizer.** Operators reselling a service to customers must join the service as an organizer. The organization service of MultiTel allows the customization of generic multimedia applications according to the organizer’s preferences and resources. The organizer may specify component or connector parameters for resource reservation, name and resource mapping, or media stream protocol selection.

- **Participant.** A simple participant can receive video and audio from the service provider (centralized applications) or from other participants.
(collaborative applications). The participant component is composed of subcomponents (InComponents) that are dynamically added according to the service logic. Each subcomponent encapsulates a specific GUI for user interaction, such as an audio or video control panel. However, the participant will only send multi-media data when the service logic allows it. This control is performed by a specific connector, which catches user input events and reports them to the other users.

Manager. The manager is a specialization of a participant component and is in charge of starting and managing organized services. In multi-party services, the manager component includes InComponents with control panels for access control, service scheduling, or resource renegotiations.

The service architecture is detailed elsewhere. Here, we focus on the multimedia and network architectures, which are closely related to resource management.

Multimedia Architecture
The compositional multimedia architecture deals with several evolutionary processes, such as changes in the hardware for multimedia applications, performance improvements, and new data representations for image, audio, video, and other media. Developers want to create applications that adapt to and take advantage of changes in platform functionality, new compression formats and so on, and that execute in heterogeneous environments. Compositional frameworks offer mechanisms for incorporating these changes into the programming environment quickly and securely. Moreover, the dynamic binding of this compositional model allows these changes to be incorporated into applications at runtime.

Figure 3 illustrates event throwing relative to the full multimedia architecture. Multimedia devices are modeled as components, while connectors encapsulate the coordination protocols that specify the dynamic relationships among components. A multimedia device (MultimediaDevice) has many nodes, where each contains a set of format components that represent the media understood by the component.

Multimedia devices compose a component hierarchy of producers, consumers, and files devices, where each component encapsulates a control panel as an InComponent object. During operation, device components generate a variety of events that could be of application interest or not. In fact, connecting components to connectors implies that the

Figure 3. Multimedia components and connectors. This partial implementation of a camera component shows the native method invocation of a node producer class loaded by the Resource Manager.
application is registering relevant events that the appropriate event handlers (connectors) will catch. Because they are platform independent, components and connectors are transferred as Java applets, except for those whose implementation depends strongly on the client system. For instance, the implementation of a multimedia device component changes with the platform, but it always throws the same events. The architecture dynamically reconfigures its multimedia devices according to the kernel of a client environment. Furthermore, nodes are InComponents that encapsulate the native code—that is, kernel-dependent programs—for controlling data reading or writing. The Resource Manager offers a standard common interface to collaborative services and manages native components and connectors. Figure 3 presents a partial implementation of a camera device (VCamera class). The prepareToSnd() method invokes the native method startVideo() defined inside the RM for camera initialization. After that, the VCamera component propagates the readyToSnd event by calling the event() method of the Component base class.

The service organizer specifies multimedia device requirements by providing a logic connection graph (LCG) as a service parameter. Then, during the resource allocation process, the local kernel translates the global parameters into specific hardware or software device parameters. The AllocCont connector controls device allocation, in accordance with the local user's resources and quality of service (QoS) requirements. The allocation is negotiated for each participant. The AllocCont extracts a participant's connection information from the global LCG, initiating the resource allocation phase. It negotiates the device's format and tries to allocate all the requested devices.

However, for service users with different levels of resource and access, three versions of this connector are defined. The first version rejects the admission of participants lacking the required devices. A second, weak version of it accepts participants independently of their resources. The third, intermediate version accepts participants having only a predefined subset of devices (for example, microphone and loudspeaker). Subsequently, designers may include participants with heterogeneous multimedia devices in the same collaborative application.

At the architectural level, MultimediaDevice is plugged into a ContCom connector that encapsulates data control commands such as Start and Pause. The network architecture, explained below, establishes connections between multimedia devices. The ContCom connector contains the STD for sender (for example, microphone), receiver (for example, loudspeaker), and sender/receiver (for example, textual chat) devices.

Figure 4 shows the sequence of actions (events and messages) for preparing user participation. The emission process begins as soon as the service architecture broadcasts an emit message to all the VirtualConnection components. The devices of the speaker's machine prepare for emission, and the rest of the conference audience prepares for data receiving. The ContCom connectors of a participant catch the send event and initiate the corresponding

Figure 4. Network architecture establishes connections between multimedia devices. Here, one user prepares to participate as a speaker and the rest of the users prepare as listeners.
The Multimedia architecture creates the connector according to the type of multimedia device component. Components subject to real-time constraints require a SynProt, instead of a ContCom, connector. The SynProt connector encapsulates the synchronization of frames delivered concurrently by two or more MultimediaDevice components.

The MultimediaSAP is a global component that encapsulates all the multimedia resources of one service session. Usually it resides on the service manager's machine because it initiates and finalizes the service. For collaborative services, in which subsets of participants maintain private side-band conference sessions, the application creates a MultimediaSAP component for representing each subsession. For instance, in an education service, the lecturer might form discussion groups represented by MultimediaSAP components that encapsulate the LCG of the private subsessions (for an example of this service, see Fuentes7).

**Network Architecture**

The network architecture defines a set of components and connectors for managing broadband multimedia channels (see Figure 5). In addition to intercomponent communication, which is resolved by the MultiTel platform, the network architecture must support the distribution of media streams. Since different services may require different transport protocols, separating the transport function from the general logic of the service affords the flexibility of building services with plug-and-play transport.

VirtualSwitch and VirtualLink base components define a common interface for making broadband stream reservations with QoS requirements. They access users' local resources by invoking the appropriate primitives implemented at the RM. For instance, services may use the RTP/RSVP protocols by merely adding the VirtualSwitch and VirtualLink components that perform RSVP resource reservations and create RTP sessions for controlling the real-time capture and reproduction of multimedia data. The control information about the packet transmission rate collected by the RTCP protocol defined inside the RTP is sent to the RResvP connector, which may modify the initial reservation of network resources. This connector encapsulates the resource renegotiation politics for each participant.

Multicast and unicast connections are modeled by specific components deriving from VirtualConnection base component type. These components define a common standard interface for setup and release transport connections. ConnectC is an external connector that controls the VirtualConnection components by mediating between multimedia components and the network platform (see Figure 5). ConnectC handles the connection setup procedures between local devices and the rest of the participants; it also handles network-initiated disconnects.

**Figure 5. Network components and connectors.** Information about users' interconnections for producer devices, such as the device role and the network protocol, is published in a HomeInfo structure.
The NetworkSAP component provides a single point of access for the service architecture, which handles the join requests of new participants and coordinates the creation and reservation of network resource components. VirtualConnection components publish the media transport information of producer devices like the transport protocol and port. This information is stored in a generic structure, HomeInfo, that may hold any kind of media stream protocol. (Figure 5 shows the HomeInfo information for a loudspeaker.)

Figure 6 shows the event sequence for adding a new participant. The joining process begins when the Multimedia architecture orders the ConnectC creation. Following this, the corresponding VirtualConnection component creates an instance and plugs into the downloaded service by adding it to the user’s USP. The ConnectC connector provides the VirtualConnection component with the necessary information for initiating the connection setup process and waits for the connectReady event. At this point, the remaining service participants must add or modify their connections. Since the NetworkSAP component encapsulates all network resources of one service, it will propagate an addUser event to every instance of the ConnectC connector, which will control its local connection setup process.

**MIDDLEWARE PLATFORM**

The middle platform includes common services related to component-connector communication, network and multimedia resource management, and application execution (see Figure 1).

**Application Directory**

The AD stores and displays the list of local and remote applications offered by the entire platform. It also facilitates access to and execution of MultiTel collaborative applications. MultiTel applications and machines may be distributed throughout the platform by hierarchical domains following the subnetting communication principle of the Internet. Each machine inside a domain is responsible for its local components, connectors, and applications. One machine per domain acts as the gateway to other domains. A routing table instructs the gateway as to the other domains to which it must forward the requests. Thus, the distribution structure can be statically or dynamically configured, the same as in IP.

For each available application, the AD includes fields for the application name, the service it offers, and general information such as the date and time scheduled to initiate the service.

\[
\text{Service} = \{ \text{Name}, \text{Description}, \text{Miscellany}, \text{state}, \text{URL}, \text{AdvanceInformation} \}
\]

The state field values can be unbound, idle, or active. Unbound services are incomplete applications that an operator-customer will reconfigure for reselling purposes; idle services are ready to be executed; and active services are those currently running on one or several machines. The AD also implements a search engine that accepts queries about application services based in any field.

The first step for launching a service is to publish it to the Web, using the AD. The initial state of the service is unbound. For example, see the TeleUni entry in Figure 7, which represents an application of remote education for a university. The organization phase starts when someone selects the service in the AD. After that, a potential operator, or organizer, must configure and register it in the idle state. In TeleUni the organizer would be a lecturer that can organize a virtual classroom; in the example of Figure 7, the class is about distributed programming.

The configuration of services involves mapping logical names to physical names and setting parameter values. We have defined the script language LCF for describing the configuration of generic services. The organization connector interprets this input configuration file and asks a customer for parameter values.

Figure 8 gives example code for the configuration of TeleUni. First, the set parties sentence states
that a videoconference is a multiparty application. After that, the organizer uses the set statement to map component and connector logical names to physical names. The USP may search a component at runtime in a location identified with a URL or in a local or remote machine. The purpose of the put statement is to set values to component parameters. This example shows the configuration of the LCG parameter of the multimediaSAP component with a filename and the QoS parameter of the networkSAP component with the best-effort value. The organizer may use the if sentence for specifying different levels of application users according to their roles, allowing heterogeneous configuration of participants.

A finished service configuration produces an output file with all parameters bound to a concrete value. The USP loads this file, which will be consulted by connectors for component-connector creation purposes.

When the service is ready for execution, the organizer initiates a session, which adds an active entry to the AD (see TeleUni-DP_1 in Figure 7). In this case, students can then enter the classroom by clicking on the active entry. The execution of the application implies the download of component code.

Resource Manager
The MultiTel Resource Manager handles multimedia and network resources. Multimedia devices and switching components are local to the client machine and therefore, unlike other service components, cannot be shared or downloaded from the network. In addition, multimedia groupware applications ideally adapt to performance differences on a given platform, such as different hardware devices, device access times, display capabilities, and resource reservation protocols. In consequence, the RM, not the USP, processes these kinds of components. The RM behaves as a resource allocator because it mediates and arbitrates conflicting requests for resources made by various parties in the system.

Because the RM creates resource components following a common interface, a generic application adapts its components dynamically to each participant’s local resources, taking into account general constraints. The RM creates software and hardware device components with a requested data format in which the hardware components are associated with a hardware device.

The RM is implemented as a Java remote object accessed by RM1. The USP object, which knows the RM’s remote reference, performs communication between the components of the application level and the RM. Although considered part of the MultiTel platform, the RM is the only component that is machine dependent. However, adding only the native code of resource components produces a version for each operating system.

Different vendors offer a variety of multimedia devices that can produce their own list of data formats. However, there is a set of typical devices that would probably be part of any multimedia service architecture, for example, a camera, speaker, video file, and so on. Bearing in mind the designer’s viewpoint, MultiTel tries to define standard interfaces and to identify the standard set of devices. Figure...
Figure 9. Multimedia components plug-ins. Hardware and software device information is stored in a MultiTel device database that the Resource Manager links to a component written in native code.

Figure 10. A logical connection graph example defining the communication technology and the list of devices that will be used during application execution.
The complete framework and platform stands at roughly 60,000 lines of Java code, including multimedia application code, but excluding C native code of local resources components. We have derived different versions of multiplayer games, video on demand, and videoconference services such as lectures and business meetings.

MultiTel offers a complete multimedia collaborative framework to support the coordination of users with heterogeneous multimedia and network resources, dynamically connecting the corresponding components and adapting them to customizable user profiles. The novel characteristics of the Java language, such as reflective programming and RMI, have facilitated the implementation of MultiTel. Unfortunately, this language poses significant drawbacks related to efficiency. This problem gets even worse when using native methods, slowing the capture and presentation of audio-video frames.

Our work now is focused on extending MultiTel with collaboration characteristics that enable the sharing of whiteboard applications. We are currently experimenting with the Java graphical packet Swing, implementing a multiuser drawing game similar to Pictionary.

**REFERENCES**


**Lidia Fuentes** is an assistant professor at the Department of Computer Science of the University of Málaga, Málaga, Spain. Fuentes received an MSc degree in computer science from the University of Málaga, Spain, in 1992, and she completed her PhD thesis on multimedia middleware, component frameworks, and telecommunication services in 1998. Her research interests address practical applications of composition and frameworks to collaborative work inside intranets, and she also works on multimedia programming in distributed systems.

**José M. Troya** is a professor in the Department of Computer Science at the University of Málaga. Troya received an MSc and PhD from the University Complutense of Madrid, Spain, in 1975 and 1980, respectively. His current research interests are coordination languages, software architectures, compositional frameworks, and heterogeneous distributed systems.

Readers may contact the authors at the Dept. of Computer Science, ETSI Informática (University of Málaga), Campus Teatinos, E29071-Málaga, Spain; email {lff, troya}@lcc.uma.es.

**CALL FOR PAPERS:**

**Survivable, High-Confidence Distributed Systems**

**Submission deadline: 12 May 1999**

**Guest Editor:** Mike Reiter, Bell Labs
reiter@research.bell-labs.com

This issue examines the state of the art in the design, implementation, and analysis of survivable distributed systems and networks. Topics include (but are not limited to) survivable networking infrastructures and routing protocols, distributed algorithms for surviving attacks on system components, and tools and middleware for simplifying the development of survivable distributed systems.

Acceptable papers can describe novel scientific advances in survivability, document experiences in developing or deploying survivable systems, or provide a survey of the state of the art in this area.

IEEE Internet Computing • Nov/Dec 1999