

Towards an Open Multimedia Service Framework

LIDIA FUENTES AND JOSÉ M. TROYA

Depto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga. E29071- Málaga (SPAIN)
<lff,troya@lcc.uma.es>

This paper describes a component-oriented framework (MultiTEL) for the development of multimedia services (MS) on top of the Web. The principal contribution of this work is the application of componentware technologies to a large-scale system. We propose a compositional model that defines the architecture of an MS as a collection of components that interact through connectors. MultiTEL architecture supports the reuse of both standard components and common interaction patterns of advanced MS. These services run on a distributed platform that supports the dynamic composition of MS components and connectors, by using Java/RMI technology.

Additional Key Words and Phrases: Compositional Frameworks, Multimedia Services, Distributed Systems, Java/Web.

A COMPONENT-ORIENTED ARCHITECTURE FOR MS

Within multimedia service platforms, many independent providers offer different kinds of services to a large population of consumers. User choice of the service provider implies that MS need to be much more responsive to evolving user requirements. This kind of competitive market and the complexity of the application domain provide appropriate conditions for the effective application of any reuse process [Schmidt and Fayad 1997].

Frameworks are of particular importance in the development of open systems, in

which the same architecture may be reused across a family of applications. Frameworks define a reusable context for components, providing a template for implementing them. We propose a compositional framework based on the component-oriented model which is presented in this section. This model has been used to design the architecture of MultiTEL.

The Composition Model

Software composition is the construction of applications from components that implement abstractions of a particular domain. Any system built from components

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its data appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 2000 ACM 00360-0300/00/0300

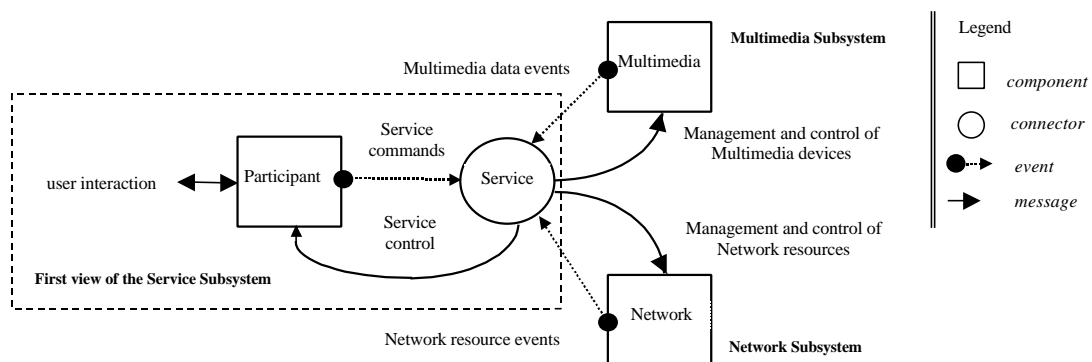


Figure 1. A first level view of MultiTEL MS architecture.

should be easy to recompose [Nierstrasz and Meijler 1995].

In particular, our component-oriented model separates the coordination patterns from data processing. Components are passive computational elements that encapsulate data, computation and/or user interaction.

When a component receives a message, it throws an event to a connector with the event primitive. However, the component does not know how the propagation of this event influences the rest of the system. This makes components independent from the different interactions they can engage in. On the other hand, connectors are abstractions of coordination patterns. A connector implements a communication protocol between two or more components and handles the events propagated by these or by other components with the catch primitive. It really knows how message exchange affects the behavior of participant components (see fig. 1) [Fuentes and Troya 1997-a].

We have tried to maximize the reusability of components and connectors by the separation of computation and coordination. Thus, a change in a protocol can be simply achieved by replacing a connector, without modifying the code of

the components involved. Likewise, changes in the internal processing of a component do not affect related connectors.

This model also incorporates a *dynamic binding* mechanism that plugs components into connectors at runtime. The dynamic linking makes the system architecture more open since there is no need to know the references of components and connectors at compile time. Section 3 explains how this mechanism works in MultiTEL.

The Architecture of the Framework

Frameworks are powerful, not just because they provide libraries of reusable components, but because they define a generic architecture that can be specialized to produce custom-built applications [Fayad and Schmidt 1997]. We have followed some key strategies to define MultiTEL architecture, such as decomposing services into standard components and defining common patterns for user interactions found in the literature [TINA-C 1995]. In order to deal with the complexity of MS we have divided the architecture into three subsystems, each one composed of domain specific components and connectors [Fuentes and Troya 1997-b].

Service subsystem. This defines the “service logic”, that is, the definition of the user

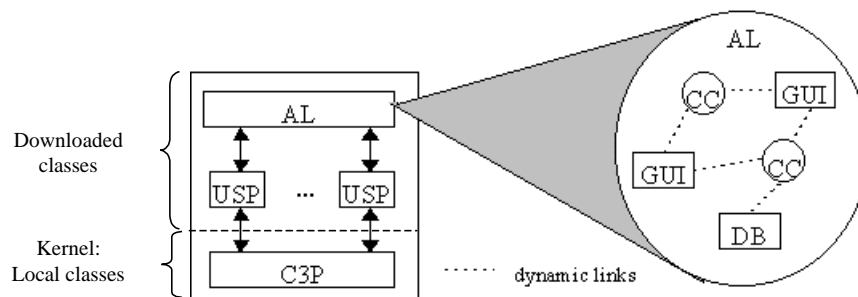


Figure 2. Java platform of MultiTel.

interaction models. Service logic is divided into many phases, each one being controlled by one connector. MultiTEL offers *Service* connectors for controlling user access, service scheduling and user-service interactions. In multi-party services the participants, arranged in a particular topology, interact according to a model of interpersonal communication. For instance, in a *Lecture* service there is only one speaker, while in a *Business meeting* service everybody can participate. In the *Lecture* service participants communicate among themselves according to a *Star* pattern. However, it is the *Full Connection* pattern which characterizes the *Business meeting* service. By just changing the scheduling connector we obtain different kinds of services. Figure 1 shows a first refinement of the service subsystem.

Multimedia subsystem. This subsystem encapsulates the management of multimedia devices, hiding their programming details from the application developer. All components modeling the same multimedia device will offer the same interface, i.e., they will throw the same events, independently from their underlying hardware. Beside this, MultiTEL provides a resource manager able to dynamically reconfigure the application components according to the user's local resources.

Network subsystem. This makes the management and reservation of broadband streams of data transparent to the rest of the system. It has components that model unicast and multicast connections between multimedia devices. These components are responsible for setting up and releasing connections on either traditional or ATM networks.

MultiTEL: AN MS FRAMEWORK IN JAVA

We have chosen Java/RMI technologies because they make the developing of interactive Web applications easier. As shown in figure 2, the implementation of MultiTEL is structured in two different levels: an object-oriented implementation of the compositional model; and a middleware platform that provides common services for the distributed execution of the applications. The first one is divided into two parts:

- *Application Level (AL).* Components and connectors of different services coexist and co-operate within this layer.
- *User-Service Part (USP).* *USPs* are persistent representations of users in a specific service. Users join a multimedia service by running the corresponding *USP* from a browser. This class deals with the dynamic binding of components and connectors, using Java/RMI for

communicating *USPs* among themselves and with the local kernel.

The second level of the framework is the middleware platform:

- *Common Component-Connector Part (C³P)*. This is the kernel of the middleware platform, in charge of looking for the appropriate components and connectors, binding them to the application’s generic references, and downloading them on execution.

Implementation of the Composition Model

MultiTEL’s base classes model components, connectors and the *dynamic binding* mechanism. One of the methods encapsulated in the *Component* class is *event()*, that implements the propagation of events (see figure 3). It is important to point out that connector method invocations do not appear in the components’ code. They only know how to delegate an event to the binding class (the *USP* class described later).

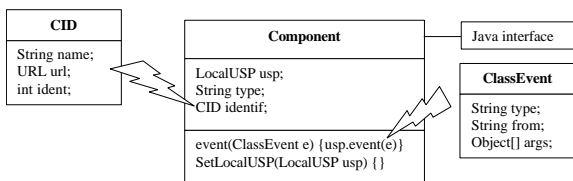


Figure 3. Component, event and Component ID (CID) classes.

Programming a new component only means inheriting this base class and implementing a Java interface that defines an MS architecture’s base type in terms of a set of public methods and a list of events. All components implementing the same

interface can propagate events to the same connectors. On the other hand, each component has a unique global address defined by its *CID* class.

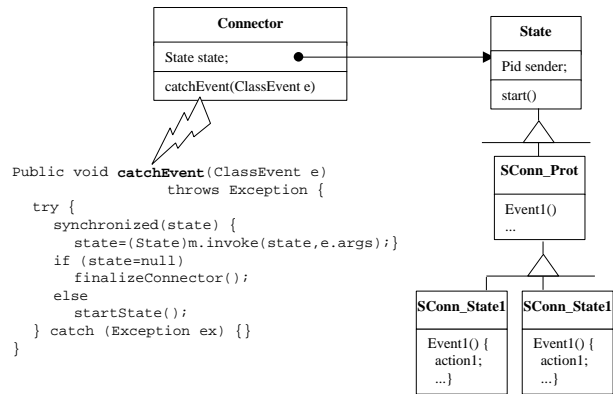


Figure 4. Connector classes. State design pattern.

The *Connector* class handles the events that trigger its state transition diagram using the *catchEvent()* method (figure 4). The implementation of the connectors’ protocols follows the *State pattern* [Gamma et al., 1995], where each state is implemented as a class with a method for each input event. These methods are directly invoked from the *catchEvent()* method by using the Reflective package of Java.

Apart from the *Component* and the *Connector* classes, the core of the framework is the *dynamic binding* mechanism that plugs the components into the connectors at runtime, using the *USP* class. The initiator of a service runs this class and downloads the general components and connectors from the machine hosting the application. This class contains information about the architecture of the application (how the components have to be plugged into the connectors), and the current application context (the components and connectors instantiated for that particular user).

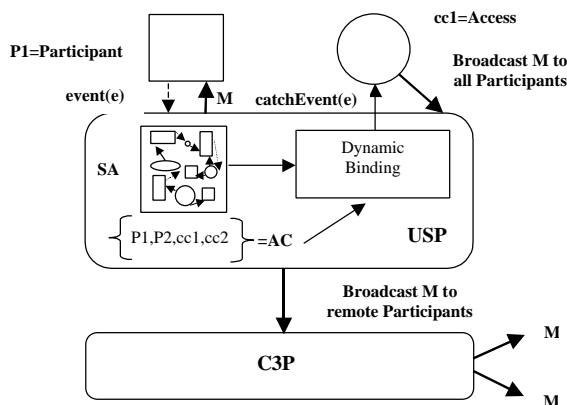


Figure 5. Dynamic binding in MultiTEL.

When a *USP* object receives a request for catching an event (*event(e)*), it looks up which connector is supposed to handle the propagated event in the current application context (*AC*) and in the application architecture (*SA*). By replacing the *USP* object, a designer can change the architecture of the application without changing its components and connectors.

In the *SA* we may also specify dynamic connections, by which several connectors can be candidates for catching an event. The appropriate connector will be placed in the *AC* according to the service logic and the customer's profile. For example, we may define a video on demand (*VoD*) service that offers different levels of access. Subscribed customers can watch complete movies while non-subscribed ones can only see the trailers. An access connector customizes this service by allocating the components and connectors that match the client's profile. A GUI component includes the control panel with *Start*, *Fwd*, *Rew*, *Pause* and *Stop* buttons. The subscribed connector enables all buttons, whereas the non-subscribed one enables the *Stop* button only.

Besides this, the framework has built-in mechanisms for broadcasting messages to a group of components. The application designer may specify the address of the target object by a *CID*, an interface name and/or a *URL*. For example, by using the second one a connector may send a message *M* to all *Participant* components, as shown in figure 5.

The Middleware Platform (*C³P*)

The *C³P* includes a Component Directory and an Application Directory. The first one is used to look for and download components and connectors, identified by their *URLs*. MultiTEL encourages the sharing of components, connectors and applications across the network. The Application Directory is a common service component that shows the list of registered applications. The customer can configure a generic service and then run, re-start or join concrete ones.

A customer of MultiTEL could be either an end-user or an organization that wants to configure a generic service for re-selling purposes. The *configuration* of a service consists of mapping logical names to physical ones and giving values to the service parameters. For example, in the *VoD* application, the 'video server' is a parameter that must be bound to a concrete *URL* address prior to execution. MultiTEL also includes a scripting language for configuring generic services.

CONCLUSIONS

In this paper we have introduced a new compositional environment for the development of *MS*, together with a framework implemented in Java that supports *MS* programming for Internet, and

the configuration of generic services. Due to the high degree of reusability obtained using MultiTEL, we have been able to significantly shorten the development times of different VoD and videoconference services, including Lectures, Conferences, and Business Meetings, which are currently the most used especially in Intranet areas. The MultiTEL environment is successfully running on Sun and Silicon Graphics workstations with Digital Media Library.

REFERENCES

FAYAD, M., SCHMIDT, D. 1997. Object-Oriented Application Frameworks. *Communications of the ACM*, Vol. 40, No. 10, pp. 32-38.

FUENTES, L., TROYA, J.M. 1997-a. Component-Oriented Service Architecture as a Flexible Service Creation Framework. *Proceedings of Communication Networks and Distributed Systems Modeling (CNDS'97) of WMC'97*. Phoenix-USA. January. pp. 68-73.

FUENTES, L., TROYA, J.M. 1997-b. A Component-Oriented Architecture to Design Multimedia Services on a Distributed Platform. *Proceedings of the Worldwide Computing and Its Applications*. LNCS, vol. 1274, August. pp. 90-105.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns*, Addison Wesley, Reading, Mass.

NIERSTRASZ, O., AND MEIJLER, T.D. 1995. Research Directions in Software Composition. *ACM Computing Surveys*, Vol. 27, No. 2, June, pp. 262-264.

SCHMIDT, D. AND FAYAD, M. 1997. Lessons Learned: Building Reusable OO Frameworks for Distributed Software. *Communications of the ACM*, Vol. 40, No. 10, pp. 85-87.

TINA-C. 1995. Overall Concepts and Principles of TINA. *Deliverable*. February.

Available at <http://www.tinac.com/>