A Component Based Distributed Software Architecture for Multimedia Services

Sakir Yucel[1] Toshihiko Kusano[2] Tuncay Saydam[1]

[1] The University of Delaware
Department of Computer and Information Sciences
Newark, DE. 19716, USA
Tel.(302)831.2716
Fax.(302)831.8458
<yucel,saydam>@cis.udel.edu

[2] NEC Corporation
1753 Shimonumabe, Nakahara-Ku Kawasaki
Kanagawa 211 Japan
Tel. +81 44 435-5512
Fax. +81 44 435-5655
kusano@ftd.trd.tmg.nec.co.jp

Abstract
In this paper, we develop a component-based, distributed software architecture to build a service and network management system for the emerging multimedia applications. This architecture is constructed by using a generic software component model. The generic software component is instantiated in order to derive specific core service and management components. User interactions as well as the inter component interactions are developed on a CORBA/JAVA platform.

Keywords: distributed software architecture, service engineering, generic software component, service management layer, CORBA, JAVA, TINA-C.

1. Introduction and Research Goals

Advances in the area of software engineering, especially the advent of object-oriented technologies, enabled the specification, design and implementation of software systems with reusable and modular components. When the emergence of different middleware solutions are added to the object and component based software engineering techniques, the developers can now design and implement the distributed object architectures transparently of the underlying enabling technologies. The service engineering also have matured to a level that we are now able to construct new value-added logical and virtual services on top of the basic network and management services. The problem, however, is to be able to integrate and inter-operate these different areas to come up with software solution for the existing problems. In the following discussion, we will describe our component-based distributed software architecture that combines the there important areas in software engineering, namely telecommunication service engineering, object-oriented technologies, and the middleware solutions. With this architecture, we propose a solution for the management of emerging multimedia services.

2. A Component-based Multimedia Service Architecture

The software architecture for multimedia service creation, access, usage, communication and management environment may be abstracted through four layers and three planes as shown in Figure 1 [SAY97]. Multimedia applications go through four layers to accomplish peer-to-peer communications: Service Management Layer (SML), Network Management Layer (NML), Element Management Layer (EML) and Network Element Layer (NEL). Each layer has its own logical and/or physical resources and software components which contribute to this communication. EML manages the physical elements that are configured for the multimedia service. NML, on the other hand, reside in the NP (Network Provider) environment and manages the network(s) resources that the end user applications will be using for the multimedia service. And finally, the SML representing the VASP (Value Added Service Provider) is involved in managing the service resources that actually provide the multimedia service. Orthogonal to these layers, we have provided three distinct planes: Service Plane, Management Plane and Communication and Control Plane. On the service plane we find the element, network and service resources and software components that are needed for the multimedia service. On the management plane, we see the management systems that manage these resources corresponding to different layers. At the intersection of planes and the layers are the software components derived from the same generic software component model.

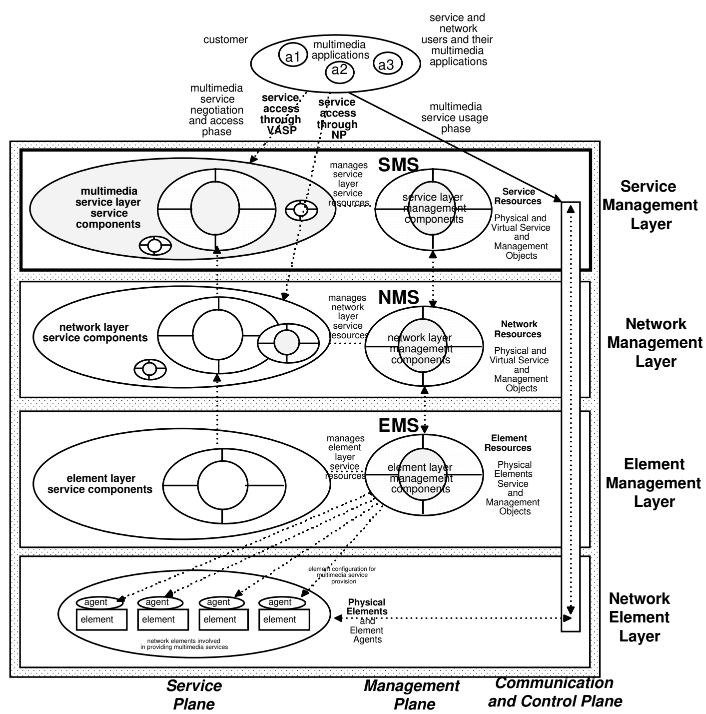

Fig 1. Multimedia Services in Four Layers and Three Planes

Communication and control plane will be used after the multimedia service is instantiated and accessed and will not be discussed in this paper. In this paper we shall mostly concentrate on the service management layer. The resources at this layer are primarily composed of multimedia service and management components and computational objects, all implemented in software. To architecture this service management layer software with the goals of software reusability, inheritance and flexibility, we will be developing a doughnut-shaped generic multimedia service component which is detailed in the following section.

3. The Generic Software Component

The generic software component is a doughnut shaped TINA-C (Telecommunications Information Networking Architecture Consortium) based component as shown in Figure 2 [YUC97],[HAI94]. It is composed of four agents and a core. The core contains a component core logic and two MIBs (Management Information Bases); the **Service_MIB,** and the **Management_MIB**. The Service_MIB contains the service parameters and objects related to the service being provided. The Management_MIB contains the resources to be monitored and managed for the service. To implement its core functionalities, that is, *what this service component does*, the component has Generic Core Logic.
Generic software component has four agents:
Usage Agent, has Usage Interface and Usage Protocol. The usage interface is a set of APIs (Application Programmer's Interface) methods that implement *what this service component offers*.
Substance Agent, has the Substance Interface. This agent provides the component to request the services of other components requesting *what this service component needs*.
Management Agent, has the management interface and management protocols. The service component is managed by this agent that relates to *how this service component is managed.*
Validation Agent. This agent with its validation interface is intended for a real-time validation (when needed) of the component.

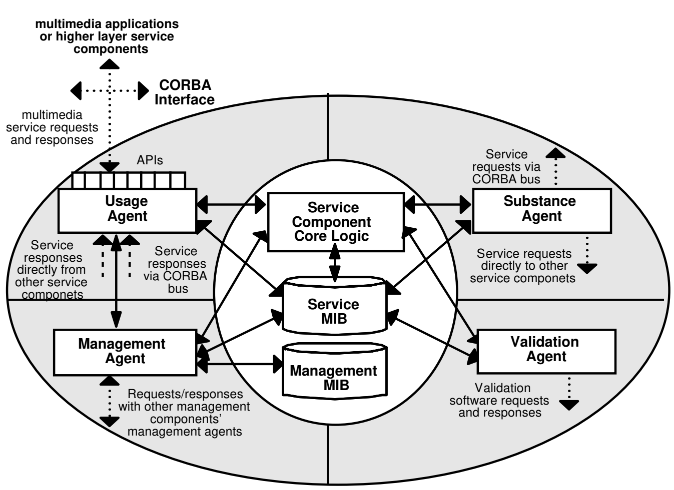

Figure 2. Generic Multimedia Service and Management Component

Armed with these powerful interfaces and with its core functionalities, the component is now ready to offer its own services, form clusters with other components to provide collection of services or even form a part of many components of a more complex service environment.

We have applied this component model to the core service components, namely, the **Connection Service Component** which provides the multimedia application with the connection setup, the **QoS (Quality of Service) Service Component** which assures the QoS parameters mapping and control, the **Security Service Component** which configures and activates the security service and mechanisms according to the security requirements of the application, and the **Bandwidth Service Component** that provides the dynamic bandwidth service. The management components corresponding to the core service components such as QoS Management Component, Connection Management Component etc. have the same generic software component structure, too. In addition of these core service and management components, the same model is utilized to evolve the **Service Interface Component**, which stands between the applications and the core service component to create, configure and activate the multimedia services as well as for the **Service Specific Component**s that contain the objects and functionality for specific application such as video conferencing, video on demand etc. Figure 3. demonstrates the organization of these software components in the SML environment.

Interactions between the user and the VASP system as well as among the software components are essential to instantiation, management, access and the usage of the multimedia services. This first type of interaction which occurs between the user application and the service interface component is explained in detail in the next section. The second type of interactions take place among the components in the service plane. This is a client/server type interaction that flows from the substance agent of the client component to the usage agent of the server component. These interactions occur over the *core service components communication channel* by calling an API function of the server component through the usage protocol.

The third type of interactions are between the same kind of components. Two directions for this kind of interaction:
1.Inter-layer communications on the same plane: This type is used for service request/response in the service plane i.e. connection service component at SML requests a service of connection service component at NML. This interaction will also use the usage protocol for calling the service APIs of the requested component. Similar interactions occur on the management plane between management components but using the management protocol CMIP. Both of these cases, the component at the higher layer is client of the component at the lower layer. The CORBA service bus between service and network layer eases client/server interactions for the components.
2.Intra-layer communications between peer level components across the planes: This type is used for service management

request/responses between a service component and its peer management through the Management Agents. This interaction uses the management protocol CMIP as well.

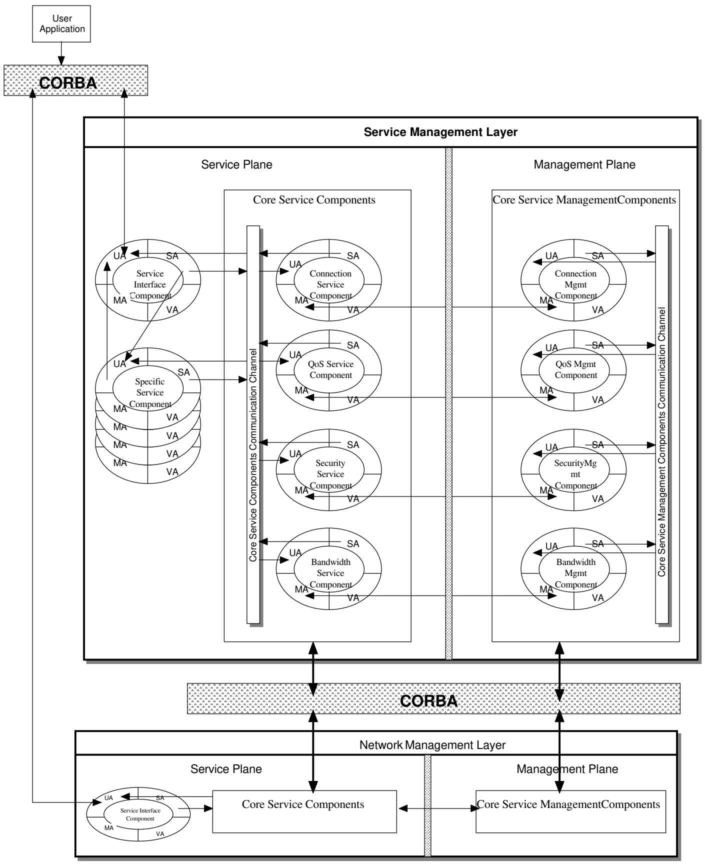

Figure3. Organization of Distributed Software Components

6. User Interactions on CORBA/JAVA Platform for Multimedia Service Creation

In this section, we shall show how we use the CORBA middleware together with JAVA language interface so that a user application accesses the services of VASP easily. In our architecture, the VASP runs an Internet Inter-ORB Protocol (IIOP) enabled server. Service Interface Component and core service components that we have discussed in previous sections are registered CORBA objects providing the interface for the customers to access, create and use the services. CORBA is a client/server distributed object infrastructure that allows the objects to locate and use the services of other objects transparently of their location, implementation language, operating system and hardware platform [WHI97]. The VASP, having the ORB server running, has an open interface to any objects willing to use it. What VASP now needs is a Graphical User Interface (GUI) that can be easily accessed so that the customers could select and configure the services that they require and could customize their environments. This is what the Java comes naturally as it generates mobile code that can move around through a special kind of application called applet, providing robust, platform independent GUI package[MAS97]. In our implementation, we use Java programming language not only for programming the user interface but also for implementing some of the service components that were discussed in previous section. Integration of JAVA and CORBA brings together the client/server paradigm, distributed object infrastructure and portable and mobile code structures which leads the current Web to the Object Web[ORF97].

The Figure 5 shows the interactions between the customer and the VASP server as well as the protocol stacks at both customer and VASP environments.

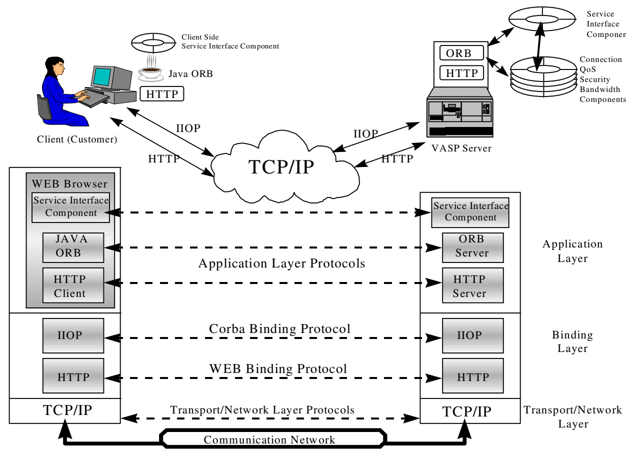

Figure 5. The Protocol Stacks at User and Service Provider Environments

The customer connects to the http server of VASP using a web browser and downloads the html page that contains the Service Creation applet using the http protocol over TCP/IP. This applet is actually the client copy of **Service Interface Component**. This Java applet is loaded into Web Browser's Java virtual machine and, having gone through a verification process, starts running in a safe environment. The applet initializes its ORB client environment. Indeed, the applet has the IDL-generated stubs that enable to connect to the particular server, which is the server copy of a Service Interface Component Object in the VASP server. From now on, the client and server applets communicate by using the usage protocol that was mentioned earlier. The binding protocol for this communication is IIOP which is the protocol to run CORBA over the Internet. The client applet, through GUI widgets, helps the customer configure the service objects at the VASP environment, and to configure the NP networks' objects that will participate for the particular multimedia service. In our implementation, we use VisiBroker for Java for ORB server and IDL/JAVA integration, since it provides both client and server side Java support, dynamic method invocation and Java mobility over IIOP.

4. Main Theoretical Results and the Contributions

This generic software component and architectures based on it have some important properties.**1.** All component, modules and services have the same internal structure as given in Figure 2. **2.** Every doughnut module or higher complexity service can be recursively derived in terms of the other triangles that it contains. **3.** A service component doughnut might partially contain other doughnuts. This way, the complex components can be created through plug-and-play of the existing components. **4**. A service component might be specialized for a given interface function or core functions. **5**. An implemented component does not have to contain all the interfaces. **6**. The way a given component is specified, designed and implemented is transparent to other service components, as long as the open standard interaction interfaces are maintained. **7**. Service components are designed and implemented to be distributed. **8.** Architecturing services in terms of the collection of generic software components promotes openness, inheritance, hierarchical structuring, separation and transparency of functionalities, as well as facilitating ease of object-oriented analysis, design and implementation. **9**. If a component, its interfaces and its information content are all specified using the same methodology and/or language, one single, seamless and coherent view will tie everything together from the network element to the service management layer. **10.** The software architecture brings together the object-oriented technologies, telecommunication management standards such as TMN (Telecommunication Management Network) and TINA-C with the middleware solutions such as CORBA, WEB and JAVA. **11.** There is a clear separation of network element layer, network layer and service layer, as well as the separation between the service and management operations. This way of service and management separation is an important contribution in the area of service engineering. This separation ensures modularity, transparency and independence in implementation of services from that of managers. **12.** Distributed component APIs are consistently developed to facilitate the component interactions. Depending on the installation, the components may be placed in the same Operating System environment, or, they may be distributed flexibly. Hence, the inter component protocol, which is the usage protocol, is designed to minimize the inter-component communication overhead. **13.** The architecture is designed to be distributed. The components at one layer communicate with the components on another layer across an open CORBA platform. **14.** The architecture can be duplicated for different services through the specific service components at the SML, such as video on demand, as well as, for different core services through the core service components, e.g. QoS service, connection service, and security service. This way of duplication integrates different kinds of core and specific services, as well as, the diversified needs of management at different layers, while, making the development of one service independent of other services. {SAY97] and [YUC97] consider the instantiation of this architecture for connection service and for security service, respectively. **15.** The architecture is dynamically configurable through plug-and-play, in that a specific service component and more importantly a core service component, new or existing, can be added or removed in the service plane as shown in Figure 3. In other words, the architecture is scalable. **16.** It also provides flexible interactions for the user through GUI over CORBA bus.

In developing our prototype, we employ an object-oriented software engineering methodology, Fusion, for the specification and design of this architecture, as it facilitates deriving the individual components from the same abstract component model nicely and reducing the complexity of the architecture. However, since our distributed software components are supposed to run asynchronously in real time with different timing and QoS constraints, part of our research concentrates on adapting Fusion for the real-time and asynchronous systems. We also believe applying the UML (Unified Modeling Language) and patterns which is presented in [LAR97] is very promising for our case.

Equally important to specifying and designing the individual components is to specify formally and implement precisely the inter component protocol. For this purpose, we will propose Estelle-like features for the life-cycle expressions in Fusion. We will then use it for specifying the interactions of system with the outside agents as well as for the specification of the inter component interaction scenarios and protocols.

We find JAVA programming language to be very natural for mapping the design object descriptions into JAVA classes, especially in terms of implementing the asynchronous and spontaneous events, threads, exceptions and timing relations. We need to develop our methodology in mapping from the design descriptions into JAVA classes to ease the implementation and the maintenance phases.

References

CAM96 Campione,M. Walrath,K. "The Java Tutorial: Object-Oriented Programming for the Internet" Addison-Wesley 1996

HAI94 Haijiang Li, "A Component Theory of Telecommunications Architectures" 2nd. Int. RACE IS&N Conf.on Intelligence in Broadband Services and Networks, September 1994

LAR97 Larman C. "Applying UML and Patterns" Prentice Hall 1998.

MAS97 Maston M.C. "Using the world wide web and Java for network service management" Integrated Network Management V, May 1997

ORF97 Orfali R. and Harkey D. "Client/Server Programming with Java and CORBA", Wiley 1997

PER95 Perrow,G.S., et al. "The Abstraction and Modelling of Management Agents" Integrated Network Management, ISINM95, May 1995

SAY97 Saydam,T."A Generic Multimedia Service and Management Architecture" Journal of Network and Systems Management, Vol.5,No.3. 1997

WHI97 Whitner R.B. "Designing scaleable applications using CORBA" Integrated Network Management V, May 1997

YUC97 Yucel, S., Kusano, T., Saydam, T. "The Generic Service Management Architecture", Proceedings Vol.1 ATIRP Conference January 1997