

White Paper: The Role of Framework Software in Value-Adding Equipment Architectures

February 28, 2006

Telecom equipment OEMs are raising the bar on where value should be added in new-product development. Increasingly, value contribution is being limited to clearly differentiating features since ROI requirements have been tightened. Even when these decisions are made without the aid of MBAs, it's tough to justify developing non-differentiating aspects of a new product when its expected market life is only three years. So, instead of the pre-1984 vertically integrated structure of "Ma Bell", telecom is emulating the value-adding structure of the computer industry.

The PC industry, and increasingly, the telecom-equipment industry, can be described as a large business ecosystem in which participants work anonymously, yet cooperatively, to develop tightly focused elements of a telecom-equipment system that can be integrated by the OEM by utilizing industry-standard interfaces. That such specialization results in increased quality, function, and performance while costs are reduced is no longer seriously debated.

This process of devolution began in 1984 with AT&T's divestiture of the regional operating companies, the IBM PC-AT, and the beginnings of computer-telephony integration (CTI). CTI was based on the use of PCs and CTI "voice boards" as the primary hardware components of enterprise systems, such as voice mail, that sold for \$20,000 and performed the same basic function of systems that incumbents were selling for five times that amount.

Ten years later the PCI Industrial Computer Manufacturers Group (PICMG) published the CompactPCI standard, which served notice that a value-adding market structure was coming to the network-equipment market as it had to the enterprise market. Now, another 10 years have passed, and PICMG is developing a new series of standards called AdvancedTCA (Advanced Telecommunications Computing Architecture). ATCA holds the promise of largely obviating the need for equipment manufacturers to invest the millions required to develop and qualify hardware that essentially performs the same functions as that of dozens of other vendors.

But what about the system-level software? It is often the largest part of a new system's development, and usually not differentiating, so why hasn't it been the subject of standardization? Protocols have and the hardware has, but what about the software? Much of the software that underlies the application, such as telephony middleware, media-processing frameworks, and signal-processing technologies, performs non-differentiating functions, and can be the subject of at least de facto standardization.

Today, there is an emerging product category of telephony framework software. There are products, such as those from Continuous Computer, GoAhead Software, and Pigeon Point Systems, that provide a framework for high-availability

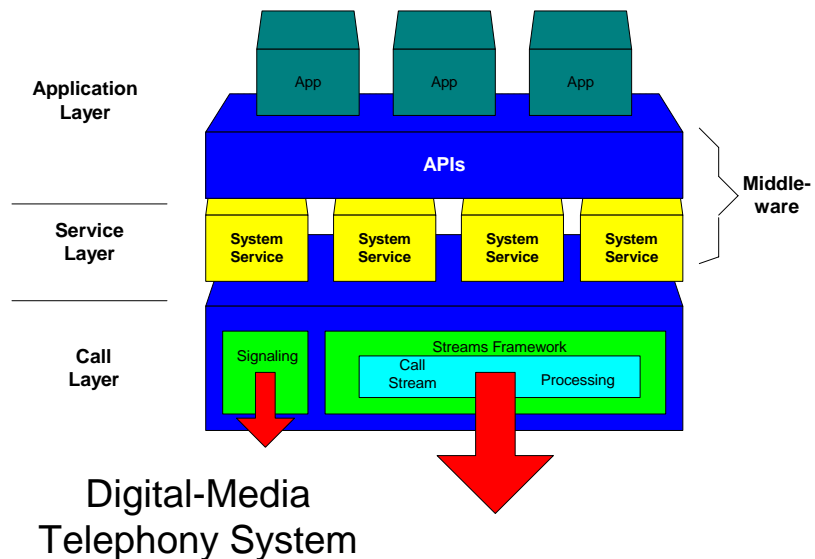


Figure 1

subsystems. It's sometimes referred to as H-A middleware. But there is also middleware to support the per-call association of a stream-processing resource with a digital-media call stream, as required in gateway and media-server systems.

In such systems, calls are placed and accepted by signaling software, such as SIP. The system then determines what processing must be performed on the call stream, and then connects the stream with the appropriate signal-processing resources. This typically requires two different frameworks: the system framework and the media-streams framework. The system framework supports the application layer, the service layer, and the attachment or integration of the signaling subsystem to the streams framework, where the actual media processing is performed, as shown in figure 1, above.

The System Framework (Telephony Middleware)

A digital-media telephony system is technically complex and will usually require a development team with a diverse skill set comprised of application, system, embedded-system, and signal-processing algorithm developers. A system's architecture should support this diversity through skill-set separation, meaning that the system should be partitioned to allow one type of developer to specify, design, and validate a major system component.

The application developer will be directly concerned with top-level system function. If the system is a gateway, what codecs are included and what signaling is supported? How are TDM calls classified? Is T.38 fax relay included? For a media server, is fax termination, audio conferencing, or IVR supported? In addition to classic PBX function, today's IP PBXs support both TDM and IP terminations, voice and fax messaging, as well as voice and T.38 fax relay transcoding functions between the two supported networks. The development of this functional range is too demanding to also burden the application developer with creating the underlying system platform. The only practicable solution is to isolate the application from the balance of the system by providing APIs such as

- System Management,
- Session Management,
- Storage Management,
- Parameter and Configuration Management,
- Call Management,
- Connection Management,
- Conferencing Management
- Resource Management,
- Terminating Voice, Fax, and Video, and
- Transcoding voice, fax, and video.

These APIs will expose function calls that support the implementation of the specified application functions, such as connect two call streams for a PBX function, play a message, send a fax, or transcode a voice stream. These commands, and the subsequent events and returns, must be sent to a software entity, a service manager (the yellow blocks in Figure 1) that encapsulates similar functions, making the system platform configurable and manageable. For example, if fax is included in a unified-messaging system, it will generally require both send and receive, and possibly image-management functions implemented in a fax-resource service manager. Voice might include play and record, indexing back and forth in a playback file, and timescale modification for speedup and slowdown. These functions would be in a voice play-record service manager.

The service managers will each have a client API. Indeed, the APIs and their supporting service managers are usually developed together by the same team of functional experts. Although the functions supported by the service manager and the API must support those required by the client application, they can be developed and validated independently of the target application's development.

How Many Applications?

Whether a system architecture supports multiple applications determines what underlying system-level facilities are required from the framework. Most of today's high-capacity integrated-media systems take advantage of multi-tasking operating systems, such as Linux, to support multiple applications. For example, an integrated enterprise communications server might have applications for PBX, voice messaging, fax, audio conferencing, IVR, and gateway functions. Such a system will require a call-routing facility (system call router or SCR) that allows applications to request call channel resources based on various criteria and to be offered inbound calls based on other criteria, such as time of day or dialed number/SIP address.

The SCR can interface with multiple call-control service managers, including, on one system, service managers for SIP, analog PSTN, and digital PSTN. The call-control service manager notifies the SCR of an offered call. The SCR selects an application by consulting configuration tables. The application may then accept or reject the call.

Resource Management

The cost of a gateway, media server, or integrated enterprise server is closely tied to the signal-processing capacity of the system. Adding a DSP blade is expensive. And adding a server blade in an HMP system, although not as expensive, is a significant portion of the total hardware cost. So how a system's media-processing resources are managed can have a significant effect on system cost.

The simplest approach is to simply avoid implementing a resource-management system by binding an application to a specific media processor (or DSP) and signal-processing software. This can be all that is required if the system has but one application that requires only one media resource, a system dedicated to fax, for example. But if the system design supports multiple applications and multiple media, a resource-management function is required for efficient resource utilization. For example, dedicating a WAN/PSTN connection and signal-processing resources to a fax application means the resources are idle unless a fax is being sent or received. An advanced resource-management facility allows the inbound fax media stream to be dynamically routed to an idle signal-processing resource where the required fax-specific resources are loaded and executed. The same task processor is used elsewhere at other times.

Such a resource-management facility is implemented by tokenizing the resources and allowing the applications to contend for them either when the application is started or on a per-call basis. Of course, the system developer must correctly configure the system to provide the required class of service for each application.

A proven approach is for the system's call router (explained above) to request the media resources required by the application to process the call and attach them to the call's media stream. This is sometimes referred to as creating a "resource group". In addition to terminating applications, resource groups can be interconnected as a part of a gateway, PBX, pre-paid calling card, or conferencing application.

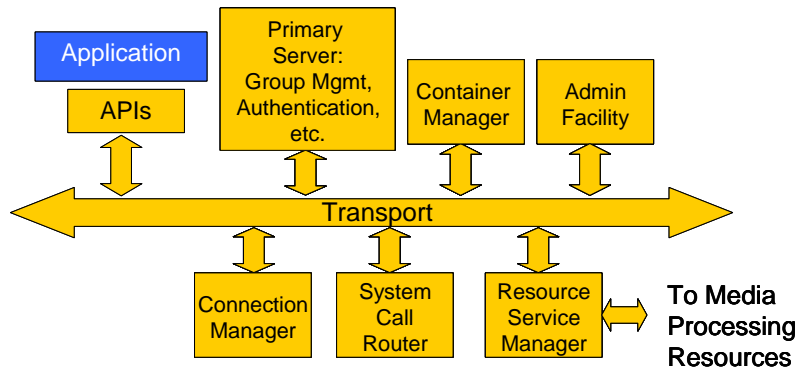
Connection Management

Today's systems often must support multiple networks, PSTN-IP gateways and integrated enterprise servers are examples. A pre-paid application may utilize both PSTN and IP networks. This means connections must be made between calls on the same switch fabric as well as heterogeneous connections. An effective connection-management facility will make these connections transparent to the application.

The application will issue a command to connect two calls, a lower-level entity within the framework will route the command to a switch-fabric controller, which will, in turn, route the command to the appropriate service manager. The connection of heterogeneous networks usually implies a transcoding operation, again, transparent to the application.

System Architecture

Since it is partitioned functionally, the system-framework architecture implied above can be implemented as either a tightly integrated system or as a distributed client-server system, delivering significant development and scalability benefits. A distributed architecture allows development of specific functions to proceed relatively independently. For example, the System Call Router can be developed and maintained independently of the Connection Manager. Effectively implemented, a client-server system can be scaled, transparent to the application, in both the client and server dimensions, simply by adding server blades in an HMP system.



Streams Framework

But that still leaves the streams framework. In Figure 2, the stream framework is used to implement the "Media Processing Resources". Indeed, in high-capacity systems efficient media-processing resource utilization is dependent on an effective multi-processor streams framework.

Figure 2 - Distributed Architecture

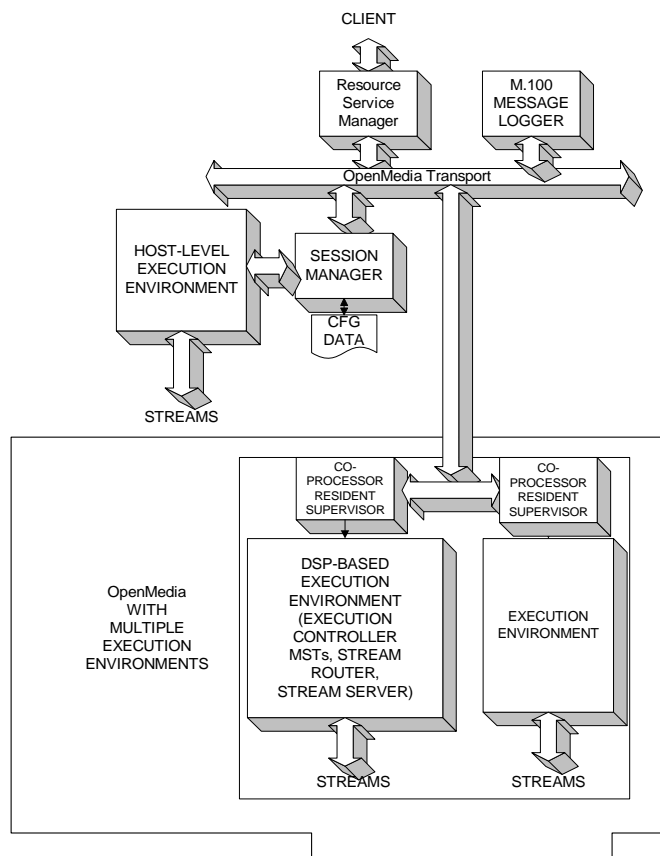
The distributed streams framework supports skill-set separation, just as does the distributed architecture of the system framework. The streams framework will receive commands from the resource service manager, typically the domain of the embedded-systems developer (see Figure 2 and the top-most block of Figure 3). It will route them to one of the multiple task processors, which can be either host- or DSP-based. Typically, algorithm developers are responsible for the task-processor software. In the example shown in Figure 3, the Session Manager handles command and event routing, isolating the determination of which of the multiple execution environments will handle the job from the service manager.

The Supervisor, in turn, shields the Session Manager from the specifics of the different execution environments. So the host-level and DSP-based environments use the same session-manager protocol, as shown in Figure 3.

The commands and returns between the Supervisor and the execution environment then cause an algorithm to be started, controlled, and stopped. The execution environment is also responsible for moving the media streams into and, if necessary, out of the environment.

Value Delivered

Resource selection, the packaging, and, of course, the application determine the function, performance, reliability and cost of a digital-media telephony



system. The decisions regarding where value is to be added—the age-old make-by decision--made by those responsible for developing the product determine the product's time to market and return on the investment in development. We have discussed the technologies behind the new product category of telephony middleware, which can have a profound influence on a new-product development project.