

White Paper - Why Do VCs Throw Money Away?

October 24, 2003

No, we're not talking about venture capitalists bidding against each other to fund the dot-com without a sensible business plan. This is about the VC that funds the digital-media telephony- equipment maker. They don't mean to throw money away; they're just unaware that much of their investment will reinvent the proverbial wheel. Just a few years ago they had no choice. But not any more. New solutions are available, and not just for early stage companies. Legacy vendors also have the same opportunity to save on new-product development while they dramatically shorten time to market and still maintain control of their strategic product platform.

Here's the case: multi-service access equipment, gateways, media servers, and enterprise communications servers are examples of digital-media systems. All share seven major architectural elements: the application software that defines the system's basic function, network interfaces and call signaling, processing resources, such as CPUs and DSPs, a box to put it all in, a system-level software framework, media-processing software, and a software framework to manage those media technologies. To the extent that any of these system elements that meet the system's requirements are on the shelf and can be acquired for less than the cost to develop them in house, and they are redeveloped anyway, money is being wasted. There may be other factors involved, such as control of the developer's product, but such concerns can be assuaged through modularity and source code. Some readers may look at the above list of architectural elements and remark that they aren't available except bundled with hardware that may not meet the OEM's requirements. And a year ago this was correct.

Generally, the OEM will develop the application since it sets the system's functionality. Off-the- shelf chassis, power supplies, and system-level boards may or may not meet the OEM's requirements, but many choices are available, including the new Advanced Telecommunications Computing Architecture (ATCA) from PICMG, which holds the promise of knocking several million dollars out of project budgets. Media technologies have been available for licensing for several years. Without that product category, integrated-media telephony systems wouldn't be economically feasible. Due to the technical complexity of today's integrated-media systems, not even the largest companies are trying to do it all.

So that leaves the software frameworks.

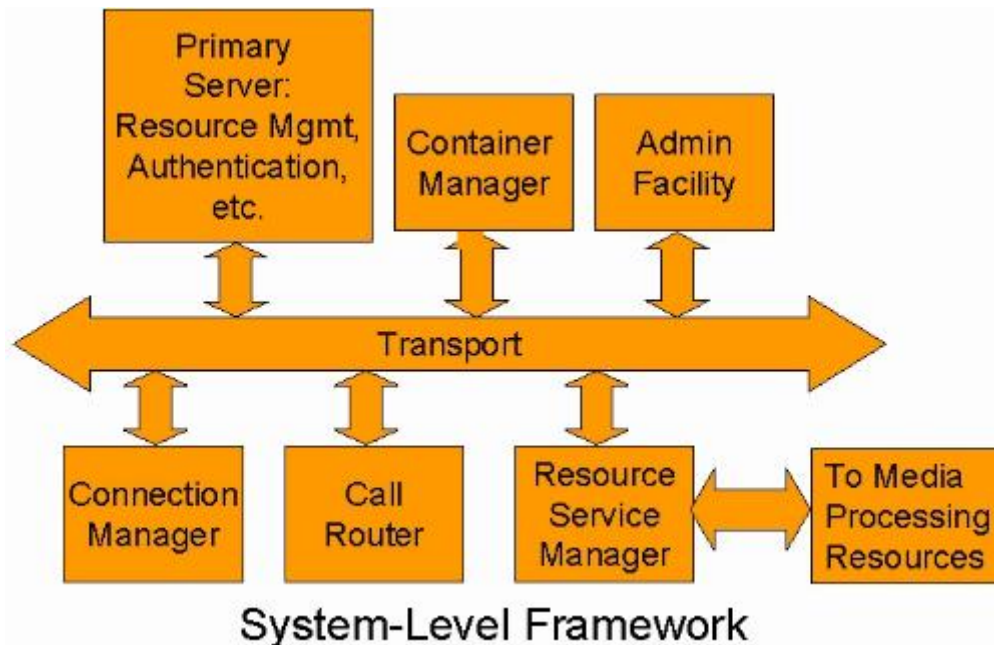
All digital-media systems have some kind of system-level software framework, often called middleware, from the simple and limited, to highly scalable client-server systems with broad functionality. But all telephony systems have some kind of framework to support application access to the media-processing technologies, which do the real work of a digital-media system. Except for the

proprietary frameworks offered by a few media-technology licensors, there are no media-streams environments, and none are standards-based and vendor-neutral, except, as will be shown, Commetrex' OpenMedia™. And, except for Commetrex' Open Telecommunications Framework® Kernel, there are, quite simply, no system-framework products for the digital-media OEM. These two products define a new product category, so not everyone will be familiar with what they do. A system-level framework handles the system-level functions common to most high-capacity digital-media systems by providing system services to the application software through APIs. These services typically include a facility for system configuration, management, and maintenance. For client-server architectures you'll usually find some kind of authentication service. For media servers, there will be a need for media storage, preferably OS non-specific. A system service that allows the application to manage call-stream connections is almost always needed. It could be PCM stream switching, packet switching/routing, or a combination. And finally, there are the media-processing resources, which are often isolated by resource service managers (RSMs).

An RSM can be designed so that it isolates the media resource from the balance of the system. The API offered to the application can be generic, allowing functionally conformant resources to be substituted without changing the application. However, this approach can lead to lowest-common-denominator resource functionality, which can be addressed by modifying the API to expose unique functionality.

A framework can be simplified by not supporting a client-server architecture, but only at the sacrifice of the extensibility and modularity inherent in the client-server architecture. Persistent media storage (e.g., "Container Manager") can be OS specific and not include media-aware functions, moving that complexity to the application and sacrificing portability. That reduces framework complexity while increasing the complexity of every application. And permanently binding media-processing and interconnection resources to an application will require less framework complexity than a system that supports dynamic resource sharing among multiple applications.

A generalization of this architecture is shown below.



Notice that the resource service manager (RSM) isolates the media-processing resources from the balance of the system. The RSM exposes high-level functions, such as "SendFax", to the application, by isolating the application from the low-level functions required to command modem data pumps and implement protocol engines, for example.

But those low-level functions must be processed by the call-stream-processing resources. In a multi-channel integrated-media system this means that call streams must be connected with the necessary transcoding software on a per-call basis. In a gateway, for example, a call on a particular PSTN interface may be voice, fax, data, or video. The system will need to classify the call and then connect the call's media stream to the appropriate media-processing software for the necessary transcoding. From there, it must be interconnected with the packet-management and network interfacing software. For a media server, a call stream resource may be connected to an announcement on one call, connected into a conference resource on another, and a fax- receive resource on yet another. This is all managed by a "streams framework", or environment, in response to commands from the application through the service layer.

A streams framework implements the system's stream-processing resources. It's where the digital-media "rubber meets the road", to paraphrase the automobile tire ad. An effective streams framework reduces the development work necessary to implement a particular media-specific resource for the system. This is done through skill-set separation, which means the electrical engineers do the media-processing (DSP) algorithm implementation and an embedded-system

developer can take care of the RSM implementation by translating high-level commands, such as play message, to lower-level functions, such as “StreamMount” and “StartGraph”. For productive development, the framework should define standardized interfaces between the system components. For example, components in the media stream will have an easy-to-use buffer-passing API.

Commetrex’ Open Telecommunications Framework® (OTF) Kernel

OTF Kernel is Commetrex’ system-level framework product that can be licensed by equipment developers interested in saving a few million dollars. OTF Kernel is

- Standards based,
- Distributed,
- Hardware independent, and includes
- Support for circuit-switched and IP transports.

OTF Kernel conforms to the ECTF S.100 recommendation (see <http://www.ectf.org>). S.100, with only one independent implementation (OTF Kernel), offers to the developer the industry’s only open application-level API and system framework. Commetrex’ OTF Kernel is a telephony middleware software product that provides telephony system services, such as authentication, resource and container management, and system management. It also normalizes stream- processing and switching resources behind resource service managers and standard client- application APIs. This means, for example, call-control can change from H.323 to SIP to ISDN, transparent to the application. Call streams are connected without regard to IP address or TDM stream:timeslot assignment. And resources, such as player-recorder and fax send-receive, are programmed the same regardless of media-resource vendor.

OTF Kernel’s RSM is the basis of its vendor and hardware independence. The OTF RSM SDK allows the licensee to add any resource to the system to perform virtually any telephony function. The SDK includes sample code for the client APIs, but the implementer is free to substitute proprietary APIs to expose proprietary functions to the client application.

OTF includes switch-fabric-specific Transport Domain Controllers (TDCs) that are responsible for implementing connection commands from the Connection Manager. The TDCs (e.g. H.100 TDC, RTP TDC) are then responsible for sending the connection command to the appropriate RSM via an open TDC-RSM protocol. The Connection Management subsystem is extensible by the OEM with the CM SDK that supports the creation of proprietary TDCs and Switch Managers.

Commetrex' OpenMedia Streams Framework

OpenMedia is Commetrex' portable streams framework. Like OTF Kernel, it is based on an open standard, the MSP Consortium's M.100 (see <http://www.msp.org>).

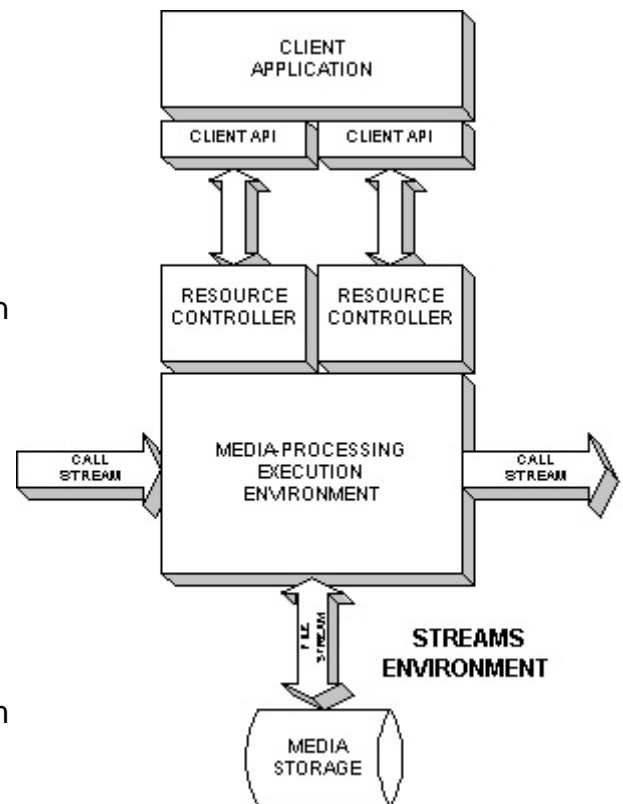
M.100 defines a software environment primarily by defining APIs with entities in the environment that control and implement media-processing system resources. The environment can exist on any stream-processing computing resource or resources, making it just as applicable to fully proprietary embedded systems or to general-purpose DSP-resource boards, such as Commetrex' MSP-H8.

M.100 allows development of M.100-compliant media-processing software or the hosting of third-party media processing software, or both. The effect is to create the same market efficiencies in developing and using multi-stream media-processing systems as DOS created for the PC back in the '80s. Specifically, the M.100 specification separates media-processing software from supporting hardware, fostering independent competition and development in these two new computer-telephony value-adding layers.

OpenMedia is a comprehensive framework supporting stream-processing software entities (Media Stream Transforms?MSTs) and the associated controlling software (the MSP Application) that delivers media-processing services to an external client.

Here 's your Get-Out-of-Jail Card

Of course, all of these system elements are included in the closed-architecture value-adding platforms of vendors such as AudioCodes, Brooktrout, and Intel. But these vendors only offer all- or-nothing integrated platforms, which include the system framework, DSP-resource boards, network interfaces, a streams framework, and media technologies. Since these products are integrated, the OEM's value add is primarily limited to developing the application. This reduces development cost and time to market, but product differentiation is limited to the application. Often that's fine, but if it's not, the OEM has no choice but to develop the hardware (unless host signal processing is used) and the two frameworks. (Remember, the media technologies are typically licensed.) But if these system elements are available as individual components, the OEM assumes control of



his product architecture. Proprietary hardware, media technologies, and framework extensions, in any combination can easily enter into the mix.

The OEM stays out of captive-technology jail and enjoys the time-to-market and development cost savings previously available only with an integrated closed-architecture platform.