Network Working Group
Internet-Draft
Expires: December 7, 2006

E. Burger
Cantata Technology, Inc.

June 5, 2006

Media Server Control Language and Protocol Thoughts
draft-burger-mscl-thoughts-01

Status of this Memo

Copyright Notice

Abstract

IP mutli-function Media Server control is a problem that has slowly bubbled up in importance over the past four years. A driver in the IETF is the requirements generated by the XCON framework. Many approaches have been proposed. Some of these proposals are device-controlled-oriented, such as H.248. Others are server-oriented, using SIP and application-oriented markup. Before rushing headlong into a framework for a solution, it is time to step back and try to understand just what the scope of the problem is. Once consensus is reached, we can then move forward with a framework for a solution.

This document describes a number of existing approaches and proposals to solve the Application Server - Media Server protocol problem, their characteristics and benefits and drawbacks.

Table of Contents

1. Introduction

An IP multi-function Media Server is a network server that provides media processing services to the network.

There are two models for media resource servers. One models the media resource server as a box of low-level resources, such as RTP mixers, transcoders, audio play and record resources, video play and record resources, tone detection and generation resources, and resources to connect, or "plumb" the resources together. The other model is that of a server that offers announcement services, interactive voice response (IVR) services (including speech recognition and speech synthesis modalities), interactive video response (IVVR) services, basic mixing services, and enhanced mixing services.

In general, when we say "multi-function Media Server", we are referring to the server model.

As the IP Media Server evolved from a box of low-level resources into a first-class server in the Internet, the protocol interfaces to control the IP Media Server evolved, as well. When people thought of the media server as a box of low-level resources, device control protocols like H.248 [1] seemed appropriate. At the time, the primary model for control of a media server was from a "SoftSwitch", or Media Gateway Controller. The principal application was for playing announcements and collecting a small number of digits. The Media Gateway Controller already implemented a device control state machine to control the Media Gateways. Moreover, the Media Gateway Controller implemented some form of Gateway Control protocol to control the Media Gateways. Thus it was logical to assume that a device control protocol, more specifically H.248 from the IETF perspective, would be appropriate for media resource control.

Although the "SoftSwitch" (traditional telephony) model (and market) was an early driver for the need for media resources, within two years it was clear that the primary consumer of media resources would be Internet-oriented applications. Developers create and deploy these applications on Internet Application Servers, using Internet and Web tools and protocols. These Application Servers have no need to control Media Gateways, and thus do not generally have implementations of device control protocols such as H.248. Moreover, Application Servers were much more likely to have HTTP [2] and SIP [3] and use stimulus-markup, client-server application architectures.

RFC3087 [4] introduced the concept of addressing services as if they were users in SIP. This meant that it was possible to address specific resources from an application simply by sending the session

to a "user" at a media server. However, RFC3087 did not provide any mechanism to achieve Internet-wide interoperability. What was needed was some sort of naming convention to address the various services available at the media server. The netann [5] specification provides such a naming convention.

Recalling the functions of a multi-function IP Media Server, the netann specification is directly sufficient for announcements and simple conferencing.

For Interactive Voice Response (IVR), VoiceXML [6] provides a standard method for defining voice (and now video) dialogs. However, there is a need to inform the IP multifunction media server that the request is for the VoiceXML service and the URI of the initial document. The netann specification provides this definition.

What is missing is a method for enhanced conference control.

By enhanced conference control, we mean facilities for creating sub-mixes, recording the mix or a leg, playing media into a mix or leg, altering the gain on a leg or the mix as a whole, defining which media is eligible for the mix, and so on.

To date, there have been several proposals, experimental protocols, and de facto standards to address the enhanced conference control problem. Factors influencing these protocols include the application's media resource model (raw resources versus service server), the desire to leverage existing protocol infrastructure (such as using SIP Registrars for resource discovery, SIP Proxies for resource location, scale, and availability), and the expectations of Internet-scale deployment sizing. The following sections examine these factors and then look at the various proposals to address them.

As a side note, two XML-based, SIP-transported media server control markup languages command approximately 100% of the market: MSCML [16] and MOML [17].

2. Factors

2.1. Media Resource Model

As the Introduction indicated, many new applications use the Internet model for media resources. That is, applications request media services from an Internet-oriented, IP multi-function Media Server. However, some legacy applications, as well as application developers more comfortable with a telco-oriented approach, would like to model the media processing function as a set of low-level resources.

There is no question that with a low-level model, one has the full flexibility to address any possible requirement. For example, creating a sidebar conference is simply the manipulation of some mixer resources and plumbing the selected RTP streams (possibly through transcoders) to the mixer resources. Likewise, one can accomplish playing a prompt to a leg by disconnecting the leg from the mixer, allocating a media player, plumbing the media player to the RTP port that represents the leg, directing the media player to play the prompt, then deallocate the media player, and finally re-plumbing the RTP stream to the mixer.

Conversely, with an Internet server model, applications request media manipulation using protocols appropriate for applications. For example, media streams are addressed using application constructs, such as SIP dialog identifiers. Rather than specifying a sidebar by manipulating RTP streams directly, the application specifies which legs the Media Server is to place into a sidebar. In fact, as we will show below, one can specify complex topologies, such as Agent/ Supervisor/Mark, with fewer messages than using a device control protocol.

2.2. Number of Protocol Messages for a Given Operation

The number of protocol messages required for a given set of operations is a factor that can potentially affect the scale of the deployment.

Too many messages can result in bandwidth problems at the media server control interface, packet handling problems at either the media server or application server, and stack processing problems at either the media server or application server.

Conversely, optimizing on number of messages can result in complex protocols with a very large number of verbs. This is often in conflict with engineering principles such as offering a simple protocol with a small number of verbs.

2.3. Network Topology

In determining the control mechanism, we need to examine the control topology. Namely, will there be a one-to-one mapping of Application Servers to Media Servers? Will there be a one-to-many mapping of Applications Servers to Media Servers? Will there be a many-to-one mapping of Applications Servers to Media Servers? Or, can there be a many-to-many mapping of Application Servers to Media Servers. Answers to this question helps determine the question as to whether there should be a single control channel per Media Server, single control channel per Application Server, single control channel per

session, or single control channel per leg.

Since control channels consume operating system resources, fewer control channels use fewer operating system resources. Of course, overall system resource utilization is more complex than simply how many channels there are at a given node. For example, on most operating systems, message routing is done in kernel space with pointer manipulation. However, once in application space, message routing is often done with buffer copying.

Another aspect influencing the cardinality of control channels is protocol layer integrity. We will examine this point in the next section.

2.4. Protocol Layer Integrity

There are many fundamental principles driving the IETF model of layered protocols. For example, a single TCP socket uses less system resources that ten thousand TCP sockets. Given that, why do we have FTP, TELNET, SMTP, NNTP, MGCP, etc.? It would appear to be much more efficient to establish a single TCP socket between the hosts and multiplex the different protocols over that socket. One of the reasons we do not do this is that while we would save on memory and kernel processing on the TCP socket, we end up spending memory and kernel processing resources on demultiplexing the TCP stream to direct the stream to the appropriate application process in user space.

Likewise, one could multiplex a given protocol over a single channel. In this case, the decision comes down to programming model. For example, in the FTP case, it is easier to manage the media and control separately over separate channels. Many implementations of FTP has the server FTP daemon spawning separate FTP server processes to handle requests. In this way the FTP server process can be quite simple and straightforward.

Another approach has multiple requests physically multiplexed to a single port, but establish separate logical sessions. One protocol that uses this model is SIP. All requests go to a single port (usually 5060), yet in the protocol data unit (PDU), we have a dialog identifier that identifies which dialog the message belongs to.

The control channel per session model maintains protocol layer integrity by allowing the kernel to do appropriate routing of requests to the application.

Multiplexing the control channel requires special considerations.

If there is a limit of a single control channel at the Media Server, then, by definition, there can be only a single Application Server controlling it. This works in a device control model, such as H.248 [1], where a Media Gateway Controller controls an entire Media Gateway. In order to allow multiple clients to control the server, one must "virtualize" the server. That is, the server presents what looks to the client as an entire, self-contained server, while in fact those self-contained servers are actually logical partitions of the physical server.

Depending on the server function, such partitioning may be easy or extremely complex. Let us consider the case of a SIP Application Server. A SIP Application Server, or Back-to-Back User Agent (B2BUA), looks to the world like a whole bunch of SIP User Agent Servers. This is not too difficult to manage, as the SIP User Agent Servers all generally look alike. On the other hand, consider a SIP Media Server. The SIP Media Server often has a fixed number of different types of resources, such as announcement players, conference bridges, recorders, and so on. Partitioning these resources can be exceedingly complex.

Some applications benefit from a single control channel model. For example, the classic SoftSwitch model and the current IMS model assume that all media processing requests go through a single network element that, in the words of TRON, is a "Master Control Program." While many from the telco world are comfortable with having a large, centralized system, many in the IETF have found time and time again that a single central server rarely meets the requirements for Internet scale. Other methods, such as server farms and alternate return contact addresses, enable theoretically infinite scale.

2.5. Computer Science Issues

Two issues to consider when using a device control protocol are how long it takes to create an application and the quality of the work product. Two factors influencing these issues are the program length and cyclomatic complexity.

There is an interesting result through 30 years of programmer productivity studies. It turns out that with the exception of the introduction of compilers, visual editors, and visual debuggers, programmer productivity has been relatively constant, at 10 to 50 lines of code delivered per day. Thus, reducing the number of lines of code required for a given function is an important tactic to achieve the goal of improving either the time-to-market or robustness of an application. This is one of the reasons why we code in Java, C++, VB, etc., instead of assembly language.

Cyclomatic complexity measures the number of branches and function calls in a given application. Again, 15 years of research have shown a strong correlation between cyclomaitc complexity and the difficulty of test and liklihood of bugs in fielded code. This is an intuitive result: more branches means more test cases, or the collary, that more branches means more code that testing will miss. However, the emperical results are more impressive: the higher the cyclomatic complexity, the more errors found in the field.

Here is a concrete example of how this plays out in practice. iSCSI [7] defines how one can, over IP, read and write blocks on a disk. One could then ask, "Why do we access data bases using data base-oriented protocols, like TDS [8]?" After all, one can do all the manipulation one needs for a data base application at the disk block level. Moreover, one can virtualize the target disk, so the application does not have to have direct control over physical disk blocks.

We would offer the answer is obvious. Data base application developers think and operate at the table access level. They don't care about disk blocks, B-Trees, indices, and so on.

One could argue that supplying a client library that hides the data base-centric operations from an application would hide the low-level nature of a disk access protocol from the application. That is, it would present an application-layer interface to the application. We offer here that protocol layer integrity comes to play here, as well. In particular, embedding data base code in the client means that one cannot have any data base innovation at the server. Everything occurs at, and is bound to, the client.

Clearly there is a need for a low-level disk access protocol. That is what drove the iSCSI effort. However, application developers need a file access protocol like NFS [10]; data base application developers need a high-level data base access protocol; mail application developers need a mail transfer protocol like SMTP [11]; and so on.

A similar situation exists in the media processing milieu. The IETF, with the ITU-T has created a media gateway control protocol, H.248 [1]. Although designed for the media gateway control problem, H.248 has capabilities for controlling arbitrary media functions, albeit at a very low level. H.248, and, THE MODEL IT REPRESENTS, assumes a master/slave, low-level device control programming model. This is analogous to direct disk block manipulation for data access, as represented by iSCSI. Features accessible via H.248 or protocols in the style of H.248 include audio players, audio recorders, RTP termination and origination, mixers, tone detectors and generators,

and plumbing primitives.

High-level media processing protocols have been proposed, modeling a media resource server as just that, a server that offers multimedia processing functions. Services offered by media servers include IVR, conference mixing, announcements, interactive video, and so on.

Consider the choice of terms: a H.248 device offers "features" while a media server offers "services". Section 3 examines the different protocol proposals in detail.

2.6. Deployment Scale

Just how many sessions do we need at any given Media Server? First, let us consider a Media Server that would handle ALL calls on the globe.

Take a population of seven billion people. Let us assume that every person calls one other person, on average, once every week. That means we are looking at 1 billion calls per day. Calculating the maximum number of simultaneous calls, let us assume that in any given populated time zone, up to 1/12th of the population of the world is actively making calls. The assumption here is that the time zones dividing the Pacific and Atlantic Oceans are essentially unpopulated (sorry Greenland and Alaska), while the time zones covering Europe have a relatively high teledensity. We make this assumption as we assume that busy hour will rotate around the Earth for a given application.

With these assumptions, there are about 83 million calls per day in a given time zone. Since, for most applications, 15% of calls occur during the busy hour, we are looking at 12.5 million simultaneous calls.

Now it is time for a reality check. Just how many simultaneous sessions will any given Application Server or Media Server really need to handle? In the above example, we found an upper limit of 12.5 million simultaneous sessions ASSUMING ALL CALLS IN THE WORLD GO THROUGH THE APPLICATION. That is a pretty hefty assumption.

What if we worked it backward? Let us assume that a single Application Server and Media Server provided voice messaging to the entire world. Again, let us start with a population of seven billion people. With a ratio of 200 subscribers per session, we get 35 million sessions. Taking time zones into account, we would be looking at about 2 million simultaneous sessions.

What is the point of these calculations? It is that the argument

that one must have a single control channel to effectively scale services is a bit disingenuous. Namely, if an Application Server will be handling, say, 100 million users, only a small percentage will be using the service at any given time. Moreover, if one architected the Application Server to be a single node, it will have to handle hundreds of thousands of inbound connections anyway. If you can handle a few hundreds of thousands of simultaneous connections, you can probably handle a few two- or three- hundreds of thousands of connections. To put this into perspective, 100,000 inbound connections represents well over 2 entire IP port address spaces.

2.7. Compatibility with SIP Model

Various proposals offer to use SIP in some way. The question is, will one use SIP within the acceptable use of SIP, or will one use it "because it is there."

For example, does a given protocol proposal leverage the SIP routing infrastructure, or is it intended for a point-to-point deployment? Does the server offer SIP-level services, or is it simply using SIP to transport, or tunnel, device control commands? Does the protocol preserve layer integrity, by using references in the SIP domain, or does it require references to the SDP [9] or IP domain?

One measure of compatibility with the SIP model a given proposal offers is to see what its compatibility with SIP Proxies, as defined by RFC3261 [3], is. For example, does the proposal require SDP manipulation? If so, how deep does the manipulation need to be? Clearly, any SDP manipulation makes the protocol incompatible with SIP Proxies - SDP modification requires the use of a back-to-back User Agent (B2BUA). Is the B2BUA simply inserting an m-line in the SDP to plumb a control channel? Is the B2BUA parsing the SDP to determine RTP addresses and media types?

The best would be pure proxies, as this will have the highest chance of avoiding compatibility issues in the future.

2.8. Security Issues

One issue is who is allowed to manipulate what at the Media Server. For services like announcements, IVR, and IVVR, a straightforward security model is to have commands come on the same SIP dialog as what established the media connection. Clearly, if you can create the connection, you have some kind of relationship with the end point, if you are not the requesting end point itself.

Other relationships get more complicated. For example, if we have a

single control pipe from the Application Server, everything is OK if there is only one Application Server. This is the model for H.248. However, if we have more than one Application Server, then we have to ensure a separation of the resources from one Application Server from another.

One solution for this problem is to partition the Media Server into multiple virtual Media Servers, each one dedicated to a given Application Server. This is a suggested model in H.248. However, as mentioned above in Section 2.4, this may be difficult for server-centric Media Servers.

3. Transport Protocols

3.1. Pure Device Control

H.248 [1] is the IETF/ITU-T media gateway control protocol. H.248 provides generic session establishment machinery and gateway internal resource interconnection. H.248 packages define various resources, including tone detectors, tone generators, audio recorders, and fixed-function audio prompt resources.

H.248 uses SDP for session negotiation, but it is considerably different than SIP's SDP offer/answer [12] protocol.

H.248 assumes a single media gateway controller per media gateway. H.248 uses a single TCP, UDP, or SCTP pipe between the controller and gateway.

Most H.248 implementations use text encoding over the wire. For those that are enamored with XML PDU's, H.248 does have an ASN.1 [13] encoding. This means one can use XER [14] to have an XML wire protocol.

3.2. Pure SIP

Using the netann [5] convention, one can perform basic media services, such as announcements and basic mixing. However, SIP does not provide the necessary controls for enhanced conferencing, such as gain control, identification of preferred speakers (if they speak, they have priority in the mix, even if they are not the loudest), creating sidebar and other topologies (such as Coach/Agent/Mark), and so on.

Note that Pure SIP uses a single TCP or SCTP socket. However, there is a separate SIP session per leg.

3.3. SIP With TCP Side Channel

MRCPv2 [15] is an example of a media processing protocol that uses a TCP side channel. In MRCPv2, the client uses SIP to route to a speech server, uses SIP's SDP offer/answer [12] protocol to negotiate the media codecs, and specifies the protocol machinery for establishing a side channel transfer protocol, such as TCP or TLS, for the actual MRCPv2 PDU's.

The MRCPv2 server hands back a unique session identifier to the client. All subsequent messages relating to a given MRCPv2 session include the session identifier. This means one can share the side channel between multiple client instances on the requesting node. MRCPv2 allows the client to request channel reuse or to request a new channel at session establishment time. Correspondingly, the MRCPv2 server can insist on a side channel per session, rather than sharing the side channel amongst sessions.

The MRCPv2 model has the benefit of using the SIP protocol machinery for session establishment. This includes using the SIP security mechanisms to authorize the association of the side channel with the media channel.

MRCPv2 itself has the drawbacks of having a totally different state machine. The MRCPv2 state machine is optimized for speech services like speech recognition and speech synthesis. Moreover, the methods are incompatible with the needs for conference control.

In addition, the MRCPv2 approach rules out the use of the protocol by SIP Proxies, as the B2BUA must modify the SDP to insert the SDP m-line for the control channel.

One might ask, "If all we are doing is establishing a TCP connection to control the media server, what do we need SIP for?" This is a reasonable question. The key is to be using SIP for media session establishment. If we are using SIP for media session establishment, then we need to ensure the URI used for session establishment resolves to the same node as the node for session control. Using the SIP routing mechanism, and having the server initiate the TCP connection back, ensures this works. For example, the URI sip: myserver.example.com may resolve to sip: server21.farm12.northeast.example.net, whereas the URI http://myserver.example.com may resolve to http://server41.httpfarm.central.example.net. That is, the host part is NOT NECESSARILY unambiguous.

3.4. SIP With INFO

Two proposals have been put forward that use the SIP dialog for the side channel. Both use the INFO method. They are MSCML [16] and MSML [18].

MSCML uses the SIP Requires and Content-Type headers to ensure interoperability and preservation of SIP semantics. MSCML correlates the commands received on the dialog with the dialog's media streams. In the case of enhanced conferences, where there are global commands such as conference size, playing to the entire conference, or recording the entire conference, MSCML has the concept of a Conference Control Leg. The Conference Control Leg is not associated with any media dialog. However, it is a SIP dialog in the normal sense.

MSML relies on a private (non-Internet) agreement between the Application Server and Media Server to know the context of the INFO messages. MSML tunnels SDP-layer information over the established dialog; in the case of media processing, it uses a secondary markup, MOML [18]. MOML is a device control protocol, with primitives similar to H.248.

Deployed versions of MOML/MSML do not use SIP, such as for referencing entities with SIP dialog properties, using SIP semantics for control, or transparently correlating SIP dialogs with RTP streams. However, the current version of the MSML specification does suggest using the SIP Dialog identifier to identify media sessions.

We will touch upon the content of what goes over the side channel in Section 4.

Using the SIP dialog for the side channel has the benefit of using the SIP routing network for getting the messages to locate and follow (in the mobility case) the UAS and UAC. In particular, proxies that are important for routing can Record-Route, while proxies that are not needed other than for session establishment can chose to not Record-Route. Thus the transport of side channel commands places only a small burden on the SIP routing network.

Note that there are a few problems resulting from the use of INFO. First, there are no throttling mechanisms, other than that provided by the underlying transport mechanism (TCP or Connection-Mode SCTP). If you are using UDP, you are out of luck. Second, even in the case of MSCML, which is well behaved in that it is guaranteed by the SIP protocol machinery that both the UAS and UAC will interoperate and understand the semantics of the MSCML INFO messages, the stacks can still get other, ill-behaved INFO messages that it may not

understand. Third, even though this has never happened in the real world, there is a theoretical problem that INFO message handling may overwhelm a proxy. In practice, one sizes ones proxies to the total traffic they need to handle. Moreover, only active element proxies, such as Edge Proxies, need Record-Route. That said, this might be a problem in the future.

The following sections explore alternatives that use the SIP Dialog.

3.5. SIP With SUBSCRIBE/NOTIFY

As outlined in the expired draft, INFO Considered Harmful [19], the events framework (SUBSCRIBE/NOTIFY) addresses all of the problems with INFO. Namely, event packages must offer throttling mechanisms, all event packages identify themselves and thus globally interoperate, and even stupid proxies that Record-Route everything often decide not to Record-Route SUBSCRIBE and NOTIFY messages.

Of course, SUBSCRIBE/NOTIFY really, really, really should not (actually, most of us, including me, say "MUST NOT") reuse the SIP dialog directly associated with the media session. This means we lose the auto-correlation feature that we have by using the INFO method.

There is a subtler, yet arguably more important problem with using SUBSCRIBE/NOTIFY. Namely, the semantics of SUBSCIBE are, "tell me (monitor) what is going on at the device." Typical uses for SUBSCRIBE are for presence [20] (what is the state of the user?), MWI [21] (what is the state of the message store?), and KPML [22] (what is the state of the key press buffer?). No package changes the state of the UAS. Using SUBSCRIBE, for example, to play a prompt or to change the configuration of a mixer, most definitely changes the state of the UAS.

3.6. SIP With MEDIA

Another approach outlined in INFO Considered Harmful [19] is to introduce a new method. This was the route taken by PUBLISH [23], as it was not quite NOTIFY.

Properly defined, a new method can safely share the SIP dialog. Moreover, it would satisfy the auto-correlation properties used by, for example, MSCML. Lastly, the semantics would be well defined, addressing the issues raised by INFO Considered Harmful.

4. Models

4.1. H.248

H.248 [1] provides:
1. A single control channel between Application Server and Media Server.
2. The possibility for an XML transport encoding.
3. Total control of media resources, at the assembly language level.

The first item is of use to those whom would want a single control channel and socket per Application Server. The second item is of use to those whom love XML. The third item ensures a measure of capabilities possibility. That is, since the Application explicitly defines the application-level semantics of media processing at the media layer, future Applications can define future, unanticipated topologies.

The drawbacks of H.248 are:
1. Layer violation et al.
2. Market adoption

The first item touches upon virtually every issue raised in Section 2. By definition, H.248 is a low-level device control protocol. That means more lines of code for a given function, higher complexity for a given function, no compatibility with the SIP model (everything becomes a MGC), and the Application Server must dive deep into SDP and they media layer to do basic operations.

The second item, while not in itself a determining factor in the IETF, is important to note as a leading indicator. For many of the reasons noted above, neither Application Server developers nor Media Server developers desire H.248 as an Application Server - Media Server protocol. Moreover, none of the major media server manufacturers have or plan to offer H.248-based media servers. In a sense, the market has spoken about this option, even in light of the 1999 declaration (well before there were any enhanced media services) by 3GPP that H.248 would be the media server (MRFP) interface.

4.2. MSCML

MSCML [16] provides:
1. Automatic correlation, including security associations, between the control channel and the media session.
2. Preservation of SIP semantics, including being SIP Proxy friendly.
3. Operations and all semantics are at the SIP dialog layer.
4. Application Servers can be relatively simple, as addressing of media processing commands is straightforward: send the command down the associated SIP media dialog.

5. Establishing a media session is straightforward: INVITE the Media Server to a session.
6. Strict adherence to the philosophy espoused by, among other places, the Application Interaction Framework [24].

The drawbacks of MSCML include:
1. Even though MSCML properly uses INFO, using INFO in itself has theoretical problems with non-interoperating devices.
2. By relying on SIP dialogs, the Application Server uses multiple SIP dialogs to control, for example, an enhanced conference on the Media Server.
3. By taking the application layer approach, MSMCL requires one to two more protocol messages than a device control approach.

The first issue is a result of using INFO.

The second issue is more interesting. For example, the enhanced conference case, that is, where one needs to play or record into the entire conference, one has to setup an additional SIP dialog, the Conference Control Dialog, per conference. In the extreme case of two-party conferences, this increases the number of SIP dialogs by 50%. Of course, few two-party scenarios require the enhanced conferencing features, and thus would not increase the number of dialogs. However, if one did need those features, then the dialog expansion would occur.

The third issue refers to the situation where the Application Server wants to place the caller into a conference, but the application needs to interact with the caller before the application knows which conference to place them into. In the MSCML model, the application has to INVITE the caller into a dialog (VoiceXML) or IVR session with the caller, determine the address of the conference, and then re-INVITE or REFER the caller into the conference.

Of course, if one uses a low-level device control markup rather than an application-level markup like VoiceXML, then the number of protocol messages to implement a voice dialog will swamp the extra redirect message.

Interestingly, MSML and MSCML exchange the same number of messages to do the same task.

The re-INVITE model offers total flexibility, in that the application never has to change if the modality of the IVR step changes. For example, the IVR step could be to a low-cost audio media resource, which then places the caller into a high-cost, 30fps, continuous presence video bridge.

The downside of the re-INVITE model is that it involves the endpoint in the SDP renegotiation. This puts an additional burden on the Application Server and caller device to relay and act upon the messages.

The REFER model does not involve the calling endpoint. However, it does have one additional protocol message.

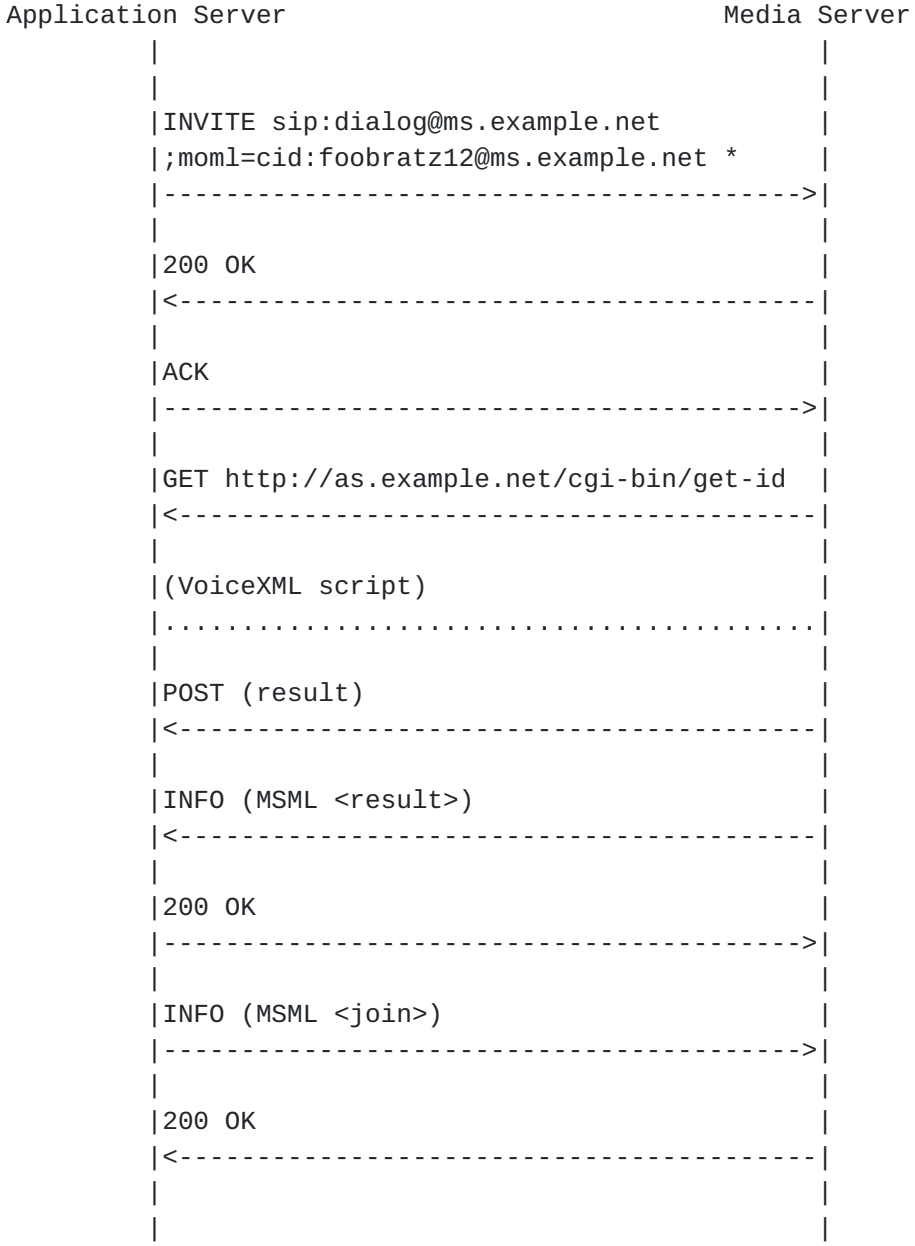
4.3. MOML/MSML

MSML [18] provides:

1. As of the -04 draft, a SIP Dialog addressing scheme.
2. Arbitrarily complex mixing topologies, on a par with H.248.
3. With MOML [17], the audio prompt, record, DTMF detection, and other functions of H.248, with the addition of access to speech resources.
4. Switching between IVR and conferencing can be done without a re-INVITE or REFER.

The drawbacks of MSML include:

1. The application has to be aware of and manipulate the media resource plumbing.
2. With most operations on a par with H.248, why not use H.248?
3. The MSML model assumes everything resides in a single server, especially with respect to the audio/video example given above.



* The MSML specification does not state how to start a session. We assume that one starts a MOML session and then send a <msml> document. The URI of the VoiceXML script, and the programming logic necessary to start that script, is embedded in the MSML document sent to the Media Server.

5. Recommendations

This section is in the spirit of getting a conversation started. Everything here is opinion. Feel free to argue.

First of all, it is clear there is interest in a standard for the Application Server - Media Server protocol in the Internet community. The adoption of MOML/MSML in the developer community and MSCML in the developer and vendor community is an existence proof of the utility of, and need for, such a protocol.

The official impetus for this work is the XCON Media Server Requirements [26]. However, in spite of the fact we have VoiceXML for application level IVR specification and H.248 for low-level IVR specification, people keep asking for IVR with conferencing, as evidenced by the XCON requirements. The problem is this IVR functionality bleeds out, and thus we need to ensure it is well thought out before just tossing something in there.

There is a desire to leverage the SIP protocol machinery for media session establishment, namely the SIP Offer/Answer protocol.

Application developers want to see the Media Server as a server that offers application-level media processing. That is, modeling the Media Server as a server that offers IVR, conference mixing, and other, application-level media processing services.

If application developers want low-level, DSP-level media manipulation, they already have an IETF protocol, H.248.

If application developers want a single control channel (total, including session establishment) from the Application Server to the Media Server, they already have an IETF protocol, H.248.

If application developers want an XML transport encoding for a low-level protocol or a single control channel, they already have an IETF protocol, H.248.

Assuming developers do not want H.248, what are the options?

INFO probably isn't it.

That leaves to directions to go. The first is to stick with the SIP Dialog model of MSCML and the other is to stick with the side channel model of MRCPv2.

The former would indicate a new method, such as MEDIA. The latter would indicate a new establishment procedure, such as described in

the other MSRP [25].

What does all this mean?

WHAT GOES DOWN THE PIPE IS AS IMPORTANT AS THE PIPE ITSELF.

It is easy to identify protocol abuse in the determination of the control channel. However, even if we have a decent control channel establishment mechanism, sending the wrong kind of messages down that channel can render the protocol less than useful.

For example, it is great to use SIP to route messages to a media server. However, if those messages emulate H.248, but encoded in XML, it would be much more efficient, cleaner, and avoid the layer violation by simply using H.248. You can even get H.248 in XML! Just please, please, please, don't transport it in SIP or a SIP side channel.

> NOTE: This is one of the reasons I pulled out of [25] at the last minute. What goes in to the pipe is as important as the pipe itself.

6. Security Considerations

One issue is who is allowed to manipulate what at the Media Server. For services like announcements, IVR, and IVVR, a straightforward security model is to have commands come on the same SIP dialog as what established the media connection. Clearly, if you can create the connection, you have some kind of relationship with the end point, if you are not the requesting end point itself.

Other relationships get more complicated. For example, if we have a single control pipe from the Application Server, everything is OK if there is only one Application Server. This is the model for H.248. However, if we have more than one Application Server, then we have to ensure a separation of the resources from one Application Server from another.

One solution for this problem is to partition the Media Server into multiple virtual Media Servers, each one dedicated to a given Application Server. This is a suggested model in H.248. However, as mentioned above in Section 2.4, this may be difficult for server-centric Media Servers.

7. IANA Considerations

As this is an Informative exploration, there are no IANA Considerations.

8. Informative References

[1] Groves, C., Pantaleo, M., Anderson, T., and T. Taylor, "Gateway Control Protocol Version 1", RFC 3525, June 2003.

[2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[3] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[4] Campbell, B. and R. Sparks, "Control of Service Context using SIP Request-URI", RFC 3087, April 2001.

[5] Burger, E., Van Dyke, J., and A. Spitzer, "Basic Network Media Services with SIP", RFC 4240, December 2005.

[6] Burnett, D., Hunt, A., McGlashan, S., Porter, B., Lucas, B., Ferrans, J., Rehor, K., Carter, J., Danielsen, P., and S. Tryphonas, "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C REC REC-voicexml20-20040316, March 2004.

[7] Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M., and E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)", RFC 3720, April 2004.

[8] Sybase, Inc., "TDS 5.0 Functional Specification Version 3.4", URL http://www.sybase.com/content/1013412/tds34.pdf, August 1999.

[9] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[10] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.

[11] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[12] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[13] Telecommunication Standardization Sector of International Telecommunication Union, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, July 2002.

[14] Telecommunication Standardization Sector of International Telecommunication Union, "ASN.1 encoding rules: XML Encoding Rules (XER)", ITU-T Recommendation X.693, December 2001.

[15] Burnett, D. and S. Shanmugham, "Media Resource Control Protocol Version 2 (MRCPv2)", draft-ietf-speechsc-mrcpv2-09 (work in progress), December 2005.

[16] Dyke, J., "Media Server Control Markup Language (MSCML) and Protocol", draft-vandyke-mscml-08 (work in progress), May 2006.

[17] Saleem, A. and G. Sharratt, "Media Objects Markup Language (MOML)", draft-melanchuk-sipping-moml-06 (work in progress), October 2005.

[18] Melanchuk, T. and G. Sharratt, "Media Sessions Markup Language (MSML)", draft-melanchuk-sipping-msml-05 (work in progress), March 2006.

[19] Rosenberg, J., "The Session Initiation Protocol (SIP) INFO Method Considered Harmful", draft-rosenberg-sip-info-harmful-00 (work in progress), January 2003.

[20] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[21] Mahy, R., "A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol (SIP)", RFC 3842, August 2004.

[22] Burger, E., "A Session Initiation Protocol (SIP) Event Package for Key Press Stimulus (KPML)", draft-ietf-sipping-kpml-07 (work in progress), December 2004.

[23] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.

[24] Rosenberg, J., "A Framework for Application Interaction in the Session Initiation Protocol (SIP)", draft-ietf-sipping-app-interaction-framework-05 (work in progress), July 2005.

[25] Boulton, C. and T. Melanchuk, "Media Server Request Protocol",

draft-boulton-media-server-control-00 (work in progress), June 2005.

[26] Even, R., "Requirements for a media server control protocol", draft-even-media-server-req-00 (work in progress), January 2005.

Appendix A. Contributors

I cannot share blame with anyone on this one.

Appendix B. Acknowledgements

Brooks Gelfand in 1985 made the quote, "If you cannot do it in assembly language, you cannot do it at all," during an argument I was having with another engineer about the relative merrits of C versus Lisp.

The catalyst for this document was the very hard and dedicated work of Chris Boulton, Tim Melanchuk, and I to bang out the and argue over the other MSRP draft, starting in April of 2005 and lasting through the very end of June.

Author's Address

Eric Burger
Cantata Technology, Inc.
18 Keewaydin Dr.
Salem, NH 03079-2839
USA

Phone: +1 603 890 7587
Fax: +1 603 457 5944
Email: eburger@cantata.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.