

:)

**subject: Media implementation in ECLIPSE I:
Principles of signaling/media separation**

date: 8 August 2000

number: HA1640000-000808-9TM

**from: Pamela Zave
FP D205
(973) 360-8676
pamela@research.att.com**

**Michael Jackson
+44 (171) 286-1814
jacksonma@acm.org**

**Eric Cheung
FP D285
(973) 360-7242
cheung@research.att.com**

TECHNICAL MEMORANDUM

Abstract

ECLIPSE is an I.P.-based platform for multimedia telecommunication services. It is an implementation of the feature-modular DFC virtual architecture, in which signaling and media processing are tightly coupled. This memorandum explains how DFC can be implemented with separate signaling and media paths, which are more efficient and more compatible with I.P. networking.

1. Introduction

Distributed Feature Composition (DFC) is a virtual architecture for the description of telecommunication services. It was designed for feature modularity, structured feature composition, and management of feature interactions [Jackson & Zave 98]. The original version has since been improved to accommodate mobile and multimedia telecommunication services [Zave & Jackson 00, Zave & Jackson 99].

DFC is the foundation of the ECLIPSE project [ECLIPSE 99]. ECLIPSE will include a service-creation environment for telecommunication services. New services described and analyzed within this environment will automatically run on the IP-based ECLIPSE network. The signaling layer of the ECLIPSE network has been

implemented and demonstrated extensively, with an *ad hoc* implementation of the voice medium [Bond *et al.* 00].

This is the first of a series of technical memoranda in which we design a general-purpose media layer for ECLIPSE. It will be the first implementation of the media aspects of DFC.

2. *Goals for media implementation*

The goals for the media layer of ECLIPSE are derived from the goals of the project as a whole [Bond *et al.* 00].

Distributability. In ECLIPSE, the signaling aspects of a feature are implemented by execution of a box program. Since this program can run anywhere in the ECLIPSE network, its location can be chosen freely in response to considerations such as efficiency and ownership. The media aspects of a feature need to be equally distributable. It must be possible to implement them anywhere in the network, even separated from the box-program execution that is controlling them.

Stability. Resource configurations, allocation strategies, and optimizations will evolve continually over time. Such resource changes must not require changes to features already installed in the ECLIPSE network—not even recompilation.

Efficiency. We use the term *media switch* to describe an implementation component with ports and the ability to switch, replicate, mute, and sum the media streams through those ports. A media switch can implement all the internal media-processing capabilities of a DFC feature box.¹ We are particularly interested in using media switches and inter-node bandwidth efficiently. Also, the post-dial setup delay for customer calls is a major concern, especially since it has been a problem in some previous implementations of IP telephony.

Media synchronization. Different media need to be synchronized within reasonable tolerances. For example, audio and video streams between two endpoints should be synchronized to within 100 ms [Jardetzky & Sreenan 95].

Fault tolerance. Fault tolerance, providing a level of reliability suitable for telecommunications, must cover both signaling and media layers of ECLIPSE.

¹In DFC, in addition to performing these functions for itself, a feature box can also connect to and use control-intensive media-processing resources [Zave & Jackson 00]. These resources have interface boxes and behave exactly like other endpoints in DFC, so they are not discussed further here.

3. *Properties of ECLIPSE*

This section documents properties of the ECLIPSE implementation of DFC that are essential to the media layer, or that have influenced the design of the media layer.

3.1. *Object communication*

The ECLIPSE software is object-oriented. For example, each instance of a DFC box program is a software object. There is no object migration, so each object resides for its entire lifetime at the network node where it was created.

Any object can send a signal to any other object, provided that the sender has the receiver's object address. We use this capability, without further comment, to implement communication between the signaling and media layers of ECLIPSE, and within the media layer of ECLIPSE.

3.2. *Channel identifiers*

The two ports of a DFC internal call can both attempt to open a media channel within that call at approximately the same time. If both choose the same channel identifier, then there is channel glare, and one of the open attempts must fail [Zave & Jackson 99].

In ECLIPSE the possibility of channel glare is eliminated by means of a convention. A DFC internal call always has a caller port and a callee port. The caller port must choose odd channel identifiers, while the callee port must choose even channel identifiers.

Another ECLIPSE convention prohibits the re-use of a channel identifier within a DFC internal call. Once a call has included the opening and closing of a channel *c*, it cannot include another `open` signal with identifier *c*. This convention prevents race conditions between old and new uses of *c*.

3.3. *Fault tolerance*

This memorandum includes the interface through which box programmers will express their media-processing

intentions and control the media layer of ECLIPSE from the signaling layer of ECLIPSE. What should happen in the case of a resource failure in the media layer?

On the one hand, the formal definition of DFC assumes a perfect—and perfectly reliable—implementation. The general philosophy of the ECLIPSE implementation is to preserve this fiction for the convenience of box programmers. Fault detection and recovery are provided as a service to the fault-tolerant main system [Bond 00].

On the other hand, it has been predicted that bandwidth shortage will be a major source of feature interaction in multimedia telecommunication systems [Tsang *et al.* 97]. If this prediction proves true in the IP context, then a few box programmers will want to receive failure reports and to program deliberate resource trade-offs to recover from them. They will not be satisfied with complete abandonment of service, which is likely to be the effect of generic recovery strategies.

The best way to encompass both viewpoints is to allow box programmers to handle exceptions when and only when they want to. In all other cases failures are reported to a fault manager, which is allowed to over-ride box programs as it engages in fault recovery.

We specify here the programming interface at the media layer. Requests for service can succeed or fail. We assume that other mechanisms not presented here distinguish the requests whose true outcome the programmer wishes to deal with. For all other requests, successes are reported to the requesting box program, while failures are reported to a fault manager which takes over from the blocked box program.

Failure of a media resource that is already in use, i.e., failures not associated with a service request, are always reported to a fault manager. We assume that no box programmer would wish to have control over these situations at the cost of having to program for asynchronous interrupts.

3.4. Nonsensical commands

It is also possible for the media layer to receive nonsensical commands—most commonly, redundant ones. When the media layer receives a nonsensical command, it protects its own integrity by ignoring the command, and keeps the usage progressing by reporting success to the box program that issued it.

To see why this is the best strategy for handling nonsensical commands, it is necessary to understand what causes them. One cause is the fact that the signaling and media layers communicate with each other, and within

themselves, using distributed protocols. As we shall see, the design of these protocols occasionally leads to redundant commands, which are not harmful or even significant.

The other cause of nonsensical commands is box-programming error. However, designing the media layer to diagnose programming errors would be a poor idea. Because the media layer does not have full information about DFC usages, it cannot detect all programming errors. Furthermore, since it would be unusual for a programmer to program explicit exception handling to deal with his own errors, runtime error detection is not very useful. Far better to rely on static analysis of box programs to eliminate errors before runtime, and to simply design the runtime environment to protect itself from residual errors.

3.5. *Media standards*

A DFC system can offer any number of media. Each medium is distinct and universal, in the sense that any sink for the medium can accept input from any source of the medium [Zave & Jackson 99].

The ECLIPSE media implementation is designed to support a relatively modest number of media, say less than ten. Considering that the only interesting media we have heard of so far are voice, video, text, images, and audio (high-quality sound), this would not appear to be a limitation.

Nevertheless, a huge proliferation of media is possible. It could arise because there are many incompatible versions of the basic media listed above. It could also arise because of the use of standards such as MPEG, in which multiple media are intertwined for reasons of synchronization and compression.

The current media implementation discourages such proliferation, because proliferation is contrary to some of the primary goals of ECLIPSE: universal interoperation, feature flexibility, and feature integration. Rather than abandon these goals, we accept the challenge of achieving adequate media performance in some other way.

4. *General approach to signaling/media separation*

The implementation of each medium is separate, at least conceptually.² So we shall describe the implementation of only one medium. Note that the only place to produce, interpret, split, merge, translate, or

²Nothing prevents the implementation of two distinct media from sharing hardware or even software, provided that the software distinguishes between the two.

otherwise bridge multiple media in a DFC system is in an interface box or a device hiding behind an interface box.

4.1. Media states in DFC

In DFC a call can have any number of bidirectional channels carrying a medium (including none). Each channel has an identifier and two *channel terminations*, one at each port of the call. A channel termination is identified by a (port,channel-identifier) pair.

In addition to the normal box ports of DFC, which are now referred to as *internal ports*, each interface box is regarded as having at least one *external port*. An *external port* is the place where a line, trunk, or resource joins the interface box. External ports enable us to specify media processing within interface boxes.

Just as ports can now be internal or external, so can channels. An *external channel* is a media channel between an external port and a device, resource, or separate network. An external channel has only one fully identified channel termination; its other termination (at a device, resource, or separate network) has only a channel identifier, and is considered *anonymous* here.³

The media-processing state of a box can be formalized as a set of *links*. Each link is a unidirectional media connection between two channel terminations; the channel terminations must have distinct ports on the same box. The links whose sink is a channel termination (p,c) specify those channel terminations at other ports of the same box whose media streams arriving as input to the box are to be summed and sent as output from the box at the channel termination (p,c) . For example, consider a box with ports p_1 , p_2 , and p_3 at which voice channels c_1 , c_2 , and c_3 respectively are open. The set of links

$$\{((p_2,c_2),(p_1,c_1)), ((p_3,c_3),(p_1,c_1)), ((p_1,c_1),(p_2,c_2))\}$$

specifies that the output from (p_1,c_1) is the sum of the inputs at (p_2,c_2) and (p_3,c_3) , the output at (p_2,c_2) is just the input at (p_1,c_1) , and there is no output at (p_3,c_3) . Thus (p_1,c_1) and (p_2,c_2) have two-way voice communication; (p_1,c_1) can also hear (p_3,c_3) , while (p_3,c_3) can hear nothing.

Henceforth pieces of DFC usages in the signaling layer will be referred to as *Dboxes*, *Dports*, *Dlinks*, *Dchannels*, and *Dcalls*. The D- prefix distinguishes them from pieces of the media layer with similar names and

³It is the intention of DFC that no external port terminate more than one channel of each medium. However, there are many potential reasons for exceptions to this rule, so we are not building it into the ECLIPSE implementation.

functions.

Figure 1 shows the Dchannels and Dlinks in a DFC usage. A Dchannel is a dashed line following a call or external line, labeled with a channel identifier. A Dlink is a dashed arrow within a box.

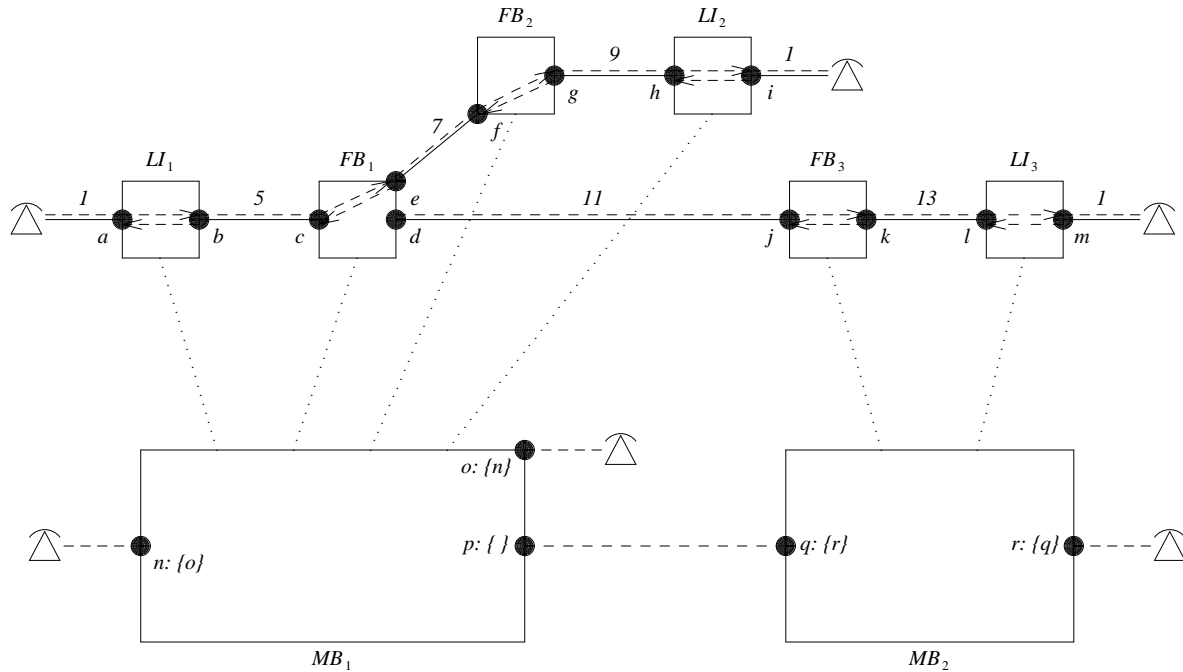


Figure 1. Dboxes in the signaling layer (top) and Mboxes in the media layer (bottom). The same devices are shown in both layers.

4.2. Mboxes

In the media layer, each media switch is encapsulated in a software component called an *Mbox*. The ports of the media switch are called *Mports*, and the bidirectional media connections between *Mports* of different *Mboxes* are named *Mcalls*. Usually we make no distinction between an *Mbox* and its media switch, as the meaning is clear from context.

For the medium whose implementation we are describing, each *Dbox* is given at the time of its creation the address of an *Mbox* of that medium. The assigned *Mbox* implements all of the *Dbox*'s processing of that medium. The *Mbox* assignment does not change during the lifetime of the *Dbox*. The dotted lines in Figure 1 show the

signaling connections between the Dboxes and the Mboxes assigned to do their media processing.

In addition to a media switch, an Mbox also includes a *controller* and a *model*. The *controller* receives and responds to commands from Dboxes. The *model* is a representation of the current media-processing states of all the Dboxes to which this Mbox has been assigned. The model contains representations of Dchannels and Dlinks, which have been created by the controller in response to Dbox commands indicating which Dchannels and Dlinks currently exist.

In addition to maintaining its model, a controller performs three functions:

- If a Dchannel connects two Dboxes with different Mboxes, then the Dchannel appears in the models of both Mboxes, and its implementation requires an Mcall between the two Mboxes. The controllers of the two Mboxes cooperate in creating and destroying the Mcalls required by their models. For example, in Figure 1 the Dchannel between $(d,11)$ and $(j,11)$ is implemented using an Mcall between Mports p and q of Mboxes MB_1 and MB_2 , respectively.
- At any instant, the media output at an Mport should be the sum of the inputs from some set of other Mports of the same Mbox. This set can be computed from the Mbox's model. It is the responsibility of the controller to keep this computation updated at all times for all Mports, and to control the media switch so that the actual outputs match the computed values. For example, in Figure 1 all Mports are labeled with the correct output sets.
- The implementation of external Dchannels requires Mcalls, for example the Mcalls from Mports n , o , and r in Figure 1. From the perspective of DFC and the model that reflects DFC, external Dchannels are opened at provisioning time and are persistent thereafter. From the perspective of the implementation, on the other hand, the Mcalls implementing them might be intermittent. It is the responsibility of the controller to bridge this gap by gathering and storing all the necessary information about an external Dchannel at provisioning time, and maintaining a corresponding Mcall at least when it is needed.

These three functions are mostly straightforward, and we do not prescribe exactly how they are performed. We merely note a few subtleties in the course of a complete explanation of Mbox models.

4.3. *Justification*

The next sections give the details of the approach we have just sketched, followed by an interim evaluation and a discussion of future work. Before plunging into the details, however, we explain what we see as the key advantages of this approach.

There is no required relationship of any kind between the configurations of Dboxes and Mboxes, either in number or location. Fewer Mboxes are usually better than more Mboxes, and it is usually best to give a Dbox an Mbox in the Dbox's own network node. But there are many good reasons for exceptions to these heuristics, and the implementors retain all degrees of freedom.

The Dbox programming interface to the media layer is determined only by DFC and the philosophy of ECLIPSE with respect to fault tolerance. It does not depend on other, less stable, characteristics of ECLIPSE such as allocation and optimization strategies (which will be implemented primarily in the media layer). The interface should remain unchanged as these strategies evolve, which would mean that box programs can also remain unchanged.

Inevitably Mboxes must execute distributed algorithms to carry out the intentions of the signaling layer. There is a potential for very complex algorithms that are difficult to understand and verify. Initially, at least, this approach requires nothing more complex than cooperation between two Mboxes to create and destroy an individual Mcall.

Finally, for all portions of a DFC usage whose Dboxes have the same Mbox, media processing is centralized and optimal.

5. *Mbox model content*

A model in an Mbox is a dynamic graph. Figure 2 is an expansion of the bottom part of Figure 1, showing the models within the Mboxes.

Nodes in the graph represent Dchannel terminations. A node on the border of the model has been allocated an Mport, while a node in the interior has not. There are two types of edge: a plain line represents a Dchannel, while an arrow represents a Dlink.

Depending on the assignment of Mboxes to Dboxes, there are two types of internal Dchannel:

- A Dchannel connecting two Dboxes, both of which have the same Mbox, for example the Dchannel between

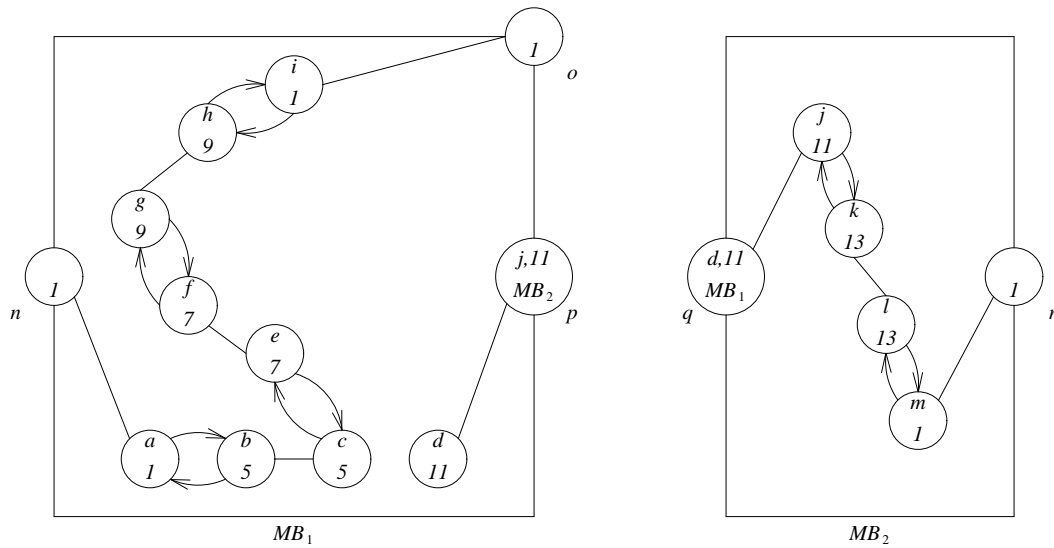


Figure 2. Mboxes in the media layer containing models. Mport allocations (n , o , p , q , r) are given to elucidate the correspondence to Figure 1.

($b,5$) and ($c,5$). The Dchannel appears in that Mbox's model. Neither of its Dchannel terminations lies on the border of the model.

- A Dchannel connecting two Dboxes, each of which has a different Mbox, for example the Dchannel between ($d,11$) and ($j,11$). The Dchannel appears in the models of both Mboxes. In each model, the remote Dchannel termination (the one whose Dbox has a different Mbox) is on the border of the model and has an Mport allocated to it. The remote Mbox is recorded in the border node's label. The Dchannel is implemented with the help of an Mcall between the two Mboxes, joining the Mports allocated to the border nodes.

In an ECLIPSE system, the models related to a particular medium are a projection of the current DFC usages onto that medium alone. This projection leaves out some important information, as mentioned in Section 3.4, most notably which Dports belong to which Dboxes. Nevertheless, the structures of the usages are still faintly visible in constraints on how models can be formed:

- A Dchannel termination can touch at most one Dchannel.
- The two terminations of a Dchannel must have distinct ports.
- The two terminations of a Dlink must have distinct ports.

- There cannot be a Dlink and a Dchannel joining the same two Dchannel terminations.

We also impose a constraint on models to ensure that there is no redundancy and no leftover garbage:

- At least one of the two Dchannel terminations of a Dlink must be touching a Dchannel.

An Mbox model is not a normal directed graph, but it can be converted to a directed graph by means of the following three-step transformation. Figure 3 shows the result of applying the transformation to the model of MB_2 in Figure 2.

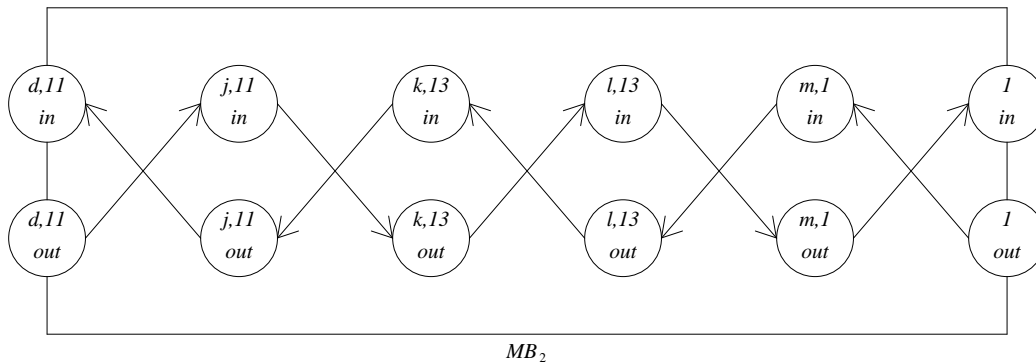


Figure 3. A directed graph corresponding to an Mbox model.

-
- (1) Replace each node with two nodes, one with the additional label *in* and one with the additional label *out*. If the Dchannel termination has a Dport, these directions should be interpreted with respect to the Dbox to which the Dport belongs. If the Dchannel termination has no Dport, these directions should be interpreted with respect to the relevant device or resource.
 - (2) Replace each Dlink from Dchannel termination t_1 to Dchannel termination t_2 by a directed edge from (t_1, in) to (t_2, out) .
 - (3) Replace each Dchannel between Dchannel termination t_1 and Dchannel termination t_2 by a directed edge from (t_1, out) to (t_2, in) and a directed edge from (t_2, out) to (t_1, in) .

The significance of the directed graph is that it defines media flow straightforwardly. At any instant, the media output at Mport m_1 should be the sum of the inputs of all the Mports m_2 such that Mport m_1 is allocated to Dchannel termination t_1 , Mport m_2 is allocated to Dchannel termination t_2 , and (t_2, in) is a predecessor of (t_1, out) .

This is well-defined even if the directed graph has cycles, which is a possibility.

It may prove appropriate for the ECLIPSE implementation to store Mbox models in the directed-graph form of Figure 3, rather than in the more compact domain-specific form of Figure 2. ECLIPSE would then be able to compute Mbox outputs using previously developed graph algorithms.

In the signaling layer, external Dchannels are created by provisioning operations. The media layer must have corresponding provisioning operations to put external Dchannels into the models of the Mboxes assigned to their interface Dboxes.

6. *Mbox model commands*

There are six commands that express all the manipulations of a particular media that a Dbox can perform by itself. These commands open and close Dchannels and Dlinks. By issuing these commands to its Mbox, the Dbox ensures that the Mbox will create representations of the correct Dchannels and Dlinks in its model, and will, in fact, implement them. Dchannel terminations in the models and Mcalls in the implementation are both manipulated as part of the semantics of these commands.

A Dbox issues a sequence of commands to its Mbox, and must wait for the reply to each command (either `ack` or `nack`) before sending another command. The sequence of commands executed by an Mbox is an interleaving of commands from the various Dboxes to which it is assigned.

For each command we discuss its arguments, which Dbox issues it, and how the Dbox gets the argument information. For each command we also give a precondition and postcondition. The precondition states what must be true for the command to be considered executable. As explained in Section 3.4, if the precondition is not satisfied then the command is ignored and acknowledged to the Dbox as if it had been successful. The postcondition states the effects of executing the command on the Mbox model and Mcalls. As explained in Section 3.3, if all these effects cannot be achieved, any partial effects are reversed, and a negative acknowledgment is sent to the Dbox or a fault manager.

The command

```
openlink(db:Dbox, from:Dchannel_termination, to:Dchannel_termination)
```

is issued by a Dbox to add a Dlink between two of its own Dchannel terminations. For the command to be executable, the Dports of the Dchannel terminations must be distinct. Also, there cannot already be a Dchannel

joining the same two Dports, or a Dlink joining the same two Dchannel terminations in the same direction. Also, at least one of the Dchannel terminations must already exist in the model.

The effect of the `openlink` command is to add to the model a Dlink from the `from` argument to the `to` argument. If one of the two Dchannel terminations does not already exist in the model, it is added.

The command

```
closeLink(db:Dbox, from:Dchannel_termination, to:Dchannel_termination)
```

is issued by a Dbox to remove a Dlink between two of its own Dchannel terminations. For the command to be executable, the Dlink must exist. The effect of the command is to remove the Dlink. Also, if this leaves a Dchannel termination in the model unconnected to anything, then that Dchannel termination is removed as well.

The commands

```
open2Link(db:Dbox, ct1:Dchannel_termination, ct2:Dchannel_termination)
close2Link(db:Dbox, ct1:Dchannel_termination, ct2:Dchannel_termination)
```

are not strictly necessary, as their effects could be achieved by `openlink` and `closeLink`, but they handle a common case more conveniently. The `open2Link` command is just like the `openlink` command except that it opens two Dlinks between the two Dchannel terminations, one in each direction (if one of the two Dlinks already exists, then the command is considered executable, but only the nonexistent Dlink is added). The `close2Link` command is just like the `closeLink` command except that it closes two Dlinks between the two Dchannel terminations, one in each direction (if only one of the two Dlinks still exists, then the command is considered executable, but only the existent Dlink is removed).

The command

```
openChan(db:Dbox, mb1:Mbox, ct1:Dchannel_termination, mb2:Mbox, ct2:Dchannel_termination)
```

is issued by a Dbox to open a Dchannel. The Dbox has received an `open` in the signaling layer (actions and constraints in the signaling layer are discussed further in Section 7). The `open` signal must be augmented with two additional arguments `mb:Mbox`, in which the sending Dbox sends its assigned Mbox for the medium being opened, and `ct:Dchannel_termination`, in which the sending Dbox sends the termination for its end of the Dchannel. The receiving Dbox assembles the `openChan` command using its own Mbox and Dchannel termination for `mb1` and `ct1`, and the received Mbox and Dchannel termination for `mb2` and `ct2`.⁴

⁴Currently the `ct` argument in the `open` signal is redundant, as the receiving Dbox could reconstruct this information by other means. We are preparing for future optimizations in which it will not be redundant.

For the `openchan` command to be executable in the Mbox that receives it, the two Dchannel terminations must have distinct Dports and the same Dchannel identifier. Also, neither Dchannel termination can already be touching a Dchannel. Also, the two Dports cannot be joined by a Dlink.

The effect of the `openchan` command in the Mbox that receives it is to add a Dchannel to the Mbox model, along with the Dchannel terminations if they are not already present. If the two Mbox arguments are the same, then neither node is a border node, and this is its only effect.

If the Mbox arguments are different, on the other hand, then there must be many other effects, some of them side-effects on the Mbox named in `mb2`. There are several ways to achieve such side-effects. Considering concerns such as ease of Dbox programming, security, and synchronization, the best approach is for Mbox `mb1` to achieve them by communicating directly with Mbox `mb2` in the media layer.

In the model of `mb1`, `ct2` must be a border node. In the model of `mb2`, the Dchannel between `ct1` and `ct2` must also appear, with `ct1` as a border node. Finally, the Mports allocated to the two border nodes must be joined by an Mcall. Mbox `mb1` does not acknowledge the `openchan` command until all of this has been accomplished successfully.

The command

```
closechan(db:Dbox,ct:Dchannel_termination)
```

is issued by a Dbox to close a Dchannel. The Dbox has received a `close` in the signaling layer at the Dport and for the Dchannel identifier combined in the argument `ct`. For the command to be executable, a Dchannel terminated at `ct` must exist.

The effect of the `closechan` command in the Mbox that receives it is to remove the Dchannel from the Mbox model, along with either Dchannel termination if it is not touching a link. Also, if any Dlink is left with both Dchannel terminations untouched by Dchannels, then the Dlink is removed. Also, if removal of a Dlink leaves a Dchannel termination untouched by Dlink or Dchannel, then the termination is removed.

If neither of the Dchannel terminations of the removed Dchannel is a border node, then these are the only effects. If one of the terminations is a border node, on the other hand, then the Mbox receiving the `closechan` command must communicate with the Mbox named in the border node to remove the same Dchannel from its model, and to destroy the Mcall associated with it. The Mbox receiving the `closechan` command does not

acknowledge it until all of this has been accomplished successfully.

Instead of issuing a single one of these six commands, a Dbox can also send a sequence of Dlink commands as a unit. The commands are executed in the order given. The sequence must succeed or fail as a unit, so if a command fails, no further commands in the sequence are attempted. Rather, the Mbox undoes the previous commands in the sequence, and sends a `nack` for the whole.

7. *Constraints on Dbox behavior*

Dbox programmers are responsible for the correctness of the media layer, and also for maintaining synchronization between the signaling and media layers. In this section we present additional constraints on which media commands Dboxes issue, and when they issue them.

A Dbox that receives an `open` signal must issue an `openchan` command, if and only if it is going to respond to the `open` with an `oack`. Furthermore, it must receive an `ack` from its Mbox for the `openchan` before responding with the `oack`. These rules help ensure that the signaling layer "thinks" there is a media channel when and only when there is one.

A Dbox that receives a `close` signal must issue a `closechan` command. Furthermore, it must receive an `ack` from its Mbox for the `closechan` before responding with the `closeack`. These rules also help ensure that the signaling layer "thinks" there is a media channel when and only when there is one.

If there is a race between two `close` signals in the signaling layer, then two `closechan` commands will be issued for the same Dchannel. Recalling that a redundant operation on an Mbox model is simply ignored and regarded as successful, this should not cause any problems.

For an `openlink` command to be executable, at least one of its Dchannel terminations must already exist in the model. A box programmer should take care that all his `openlink` commands are issued only when they are executable.

Apart from this, a Dbox has no built-in obligations whatsoever concerning Dlink commands. It simply issues the Dlink commands it needs to perform its function. In particular, the model constraints and command pre/postconditions are designed so that no garbage is left over if a Dbox dies with Dlinks left in the model.

In addition to having the media layer be correct, we would also like it to be fast. This also can be influenced

by Dbox programming. We shall illustrate the point by showing a fast form of transparent media behavior, in which a Dbox continues a media channel without attempting to affect it in any way.⁵

The key issue in this example is easy to see from the rule "A Dbox that receives an open signal must issue an openchan command, if and only if it is going to respond to the open with an oack." How does a transparent Dbox know how it is going to respond to its incoming open until it receives a response to its outgoing open from downstream? For purposes of faster media setup, the Dbox can assume that the media channel will be required, and eagerly issue the openchan before knowing this for sure.

We consider a Dbox j with callee port p_1 and caller port p_2 . The successful case is shown in the message-sequence chart of Figure 4, where Dbox j is a process along with its predecessor in the usage $j-1$ and its successor in the usage $j+1$. There is also a process for the Mbox m assigned to all three Dboxes (the sharing is just for convenience—the point of this example would not change if the Dboxes had different Mboxes).

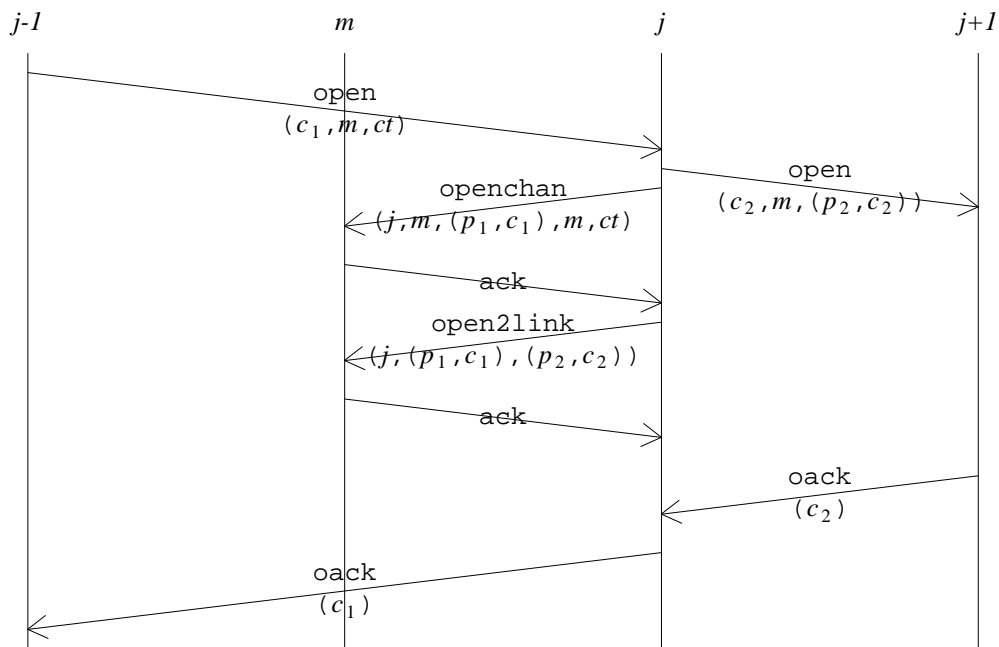


Figure 4. A fast way of achieving transparent media behavior.

In Figure 4 the original open arrives at port p_1 of j with channel identifier c_1 . Dbox j immediately

⁵In this example, we do not consider the possibility of resource failures. If a Dbox is behaving transparently with respect to a media channel, it will definitely be delegating this form of exception handling to a fault manager.

propagates the `open`, choosing channel identifier c_2 to prevent glare. Then j immediately issues the two media commands to implement the section of the media channel that it controls. In doing this, it is assuming that it is going to respond to its incoming `open` with an `oack`. The assumption proves correct, because its outgoing `open` receives an `oack` in return. Dbox j does not send its `oack` upstream until it has received both an `oack` from downstream in the usage and an `ack` from m for its final media command.

Figure 5 shows what must happen when the eager, optimistic assumption proves false. On receiving `onack` from downstream, Dbox j first propagates it upstream to minimize end-to-end signaling delay. Dbox j then issues a `closechan` command to close the Dchannel that it erroneously opened. Note that it is not necessary for j to issue a `close2link` command, because the closing of the Dchannel will remove the Dlinks from the model automatically. Note also that it is safe to send the `onack` before sending the `closechan` or receiving an `ack` of it, because there is no possibility that an end-to-end media channel can exist, even transiently.

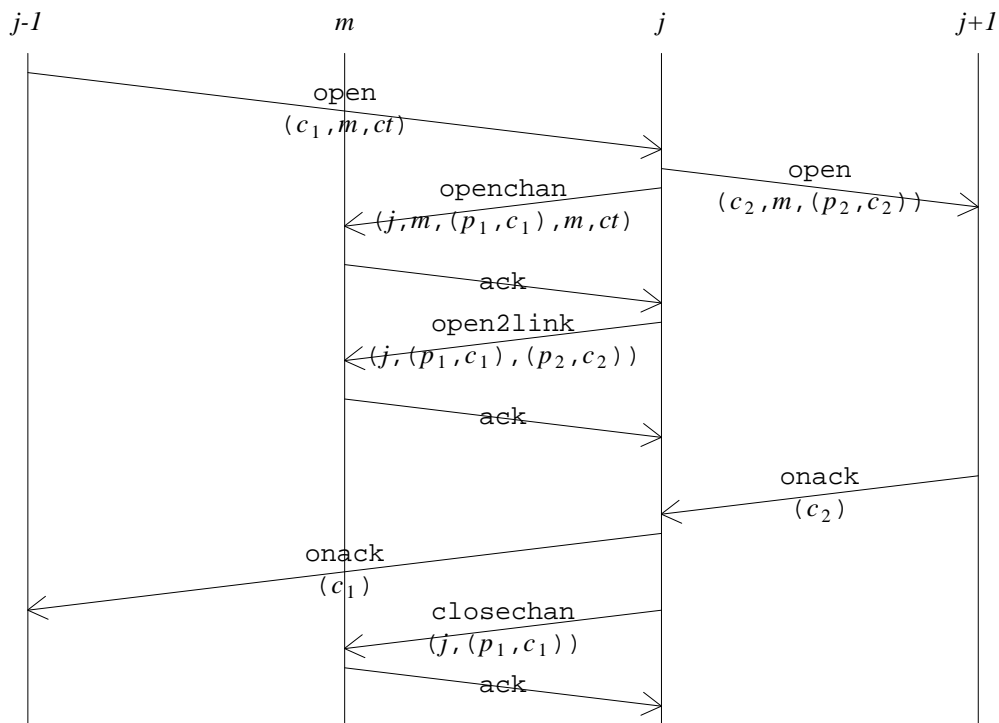


Figure 5. Recovery after a fast way of achieving transparent media behavior goes wrong.

8. *A modest optimization*

In Figure 1, the input media stream at Mport p is not being used for anything, as can be seen from the fact that p does not appear in the output set of any Mport of MB_1 . In this case, MB_1 can signal to MB_2 to stop transmitting from its Mport q (recall that the identifier of MB_2 is stored in the Dchannel termination to which p is allocated). When the situation changes and the input at p is needed once again, MB_1 can signal to MB_2 to start transmitting. This simple optimization conserves inter-node bandwidth. A mute/unmute signal is ignored if it arrives after the associated Dchannel has been closed.

9. *Interim evaluation and future work*

Distributability. The media implementation of a feature is completely distributable—it can be located anywhere in the network. The only limitation is that the Mbox used by a Dbox must be chosen when the Dbox is created, and cannot be changed afterward.

Stability. Once the initial plan is completed (see also [Cheung et al. 00]), Dbox programming will be well-insulated from changes to ECLIPSE.

Efficiency. Media switches are not constrained by this scheme, so they can be deployed across the network efficiently.

The technique illustrated in Figures 4 and 5 allows segments of the media path to be set up in parallel with each other and with some of the signaling. We hope that, as a result, post-dial setup delay will be determined mainly by delay in the signaling layer. Delay in the signaling layer is a significant concern to the ECLIPSE project, so it is important that the media layer not make the problem worse.

Concerning the important issue of inter-node bandwidth, there is still much work to be done. For one thing, the efficiency of this scheme depends a great deal on how Mboxes are assigned to Dboxes. In some cases it is obvious—line interfaces and feature boxes in the source and target zones of their addresses should all be given Mboxes on the line's access node—but in many other cases it is not. We need a plan for the harder cases, such as feature boxes subscribed to by mobile addresses. Until we have such a plan, we will not know how serious is the limitation that the Mbox used by a Dbox must be chosen when the Dbox is created.⁶

⁶A piece of good news: At the time a free Dbox is created, the Mbox assignments of other Dboxes already in the usage can be taken into account.

However the Mbox allocation goes, it is clearly the case that Mports, inter-node bandwidth, and other resources can be wasted by hairpins. Future work will show how to avoid or remove hairpins in many cases [Cheung et al. 00].

Media synchronization. We have done nothing so far, and can only hope that the approach presented here proves compatible with this requirement.

Fault tolerance. A strategy for handling media-layer resource failures is already integrated into this work. We still need to explore other dimensions of fault tolerance, for example the effects of node failure. Failure of a node might cause a portion of a usage to malfunction, with as-yet-unknown effects on both the signaling and media layers.

In another vein, we are also concerned about errors in the algorithms described here. We intend to use the Alloy Constraint Analyzer [Jackson 00] to verify that the pre- and postconditions of the model operations are necessary and sufficient to preserve the model constraints. In addition, there are many other model properties to explore. For example, it seems that every path through the directed-graph form of a model (Figure 3) should have a strict alternation between edges derived from Dchannels and edges derived from Dlinks. As we explore further and further from the core constraints, it becomes harder and more important to understand exactly where responsibility for preserving a desirable property lies.

10. Acknowledgments

These results would not have been possible without our colleagues in the ECLIPSE project Greg Bond, Hal Purdy, Chris Ramming, and Xiaotao Wu.

11. References

[Bond 00]

Gregory W. Bond. Fault management issues for a next-generation IP telecom services architecture. In *Workshops and Abstracts of the International Conference on Dependable Systems and Networks*, pages D-11 - D-13. IEEE Computer Society Press, June 2000.

[Bond et al. 00]

Greg Bond, Eric Cheung, Michael Jackson, Hal Purdy, Chris Ramming, and Pamela Zave. Investigations of a new service architecture for the next generation of telecommunications. Submitted for publication, April

2000.

[Cheung et al. 00]

Eric Cheung, Michael Jackson, and Pamela Zave. Media implementation in ECLIPSE II: Transparent behavior and its optimizations. AT&T Technical Memorandum in preparation.

[ECLIPSE 99]

ECLIPSE stands for "Extended Communications Layered on I.P. • Synthesis Environment". See our web site at <http://eclipse.research.att.com>.

[Jackson 00]

Daniel Jackson. Alloy Constraint Analyzer web site at <http://sdg.lcs.mit.edu/alloy>.

[Jackson & Zave 98]

Michael Jackson and Pamela Zave. Distributed feature composition: A virtual architecture for telecommunication services. *IEEE Transactions on Software Engineering* XXIV(10):831-847, October 1998.

[Jardetzky & Sreenan 95]

P. Jardetzky and C. Sreenan. Storage and synchronisation for distributed continuous media. *Multimedia Systems* III(3):151-161, September 1995.

[Tsang et al. 97]

Simon Tsang, Evan H. Magill, and Bryce Kelly. The feature interaction problem in networked multimedia services—present and future. *British Telecom Technology Journal* XV(1):235-246, January 1997.

[Zave & Jackson 99]

Pamela Zave and Michael Jackson. DFC modifications II: Protocol extensions. AT&T Technical Memorandum HA1640000-991119-16TM.

[Zave & Jackson 00]

Pamela Zave and Michael Jackson. DFC modifications I (Version 2): Routing extensions. AT&T Technical Memorandum HA1640000-000128-4TM.