Inter-Office Memorandum

DRAFT - DRAFT - DRAFT - DRAFT

To Pilot interest

Date November 7, 1977

From Paul McJones

Location Palo Alto

Subject Adding new devices to the Channel interface

Organization SDD/SD

XEROX

Filed on: [Ifs]<McJones>Channel.memo

The problem

The Channel interface in the second draft of the Pilot Functional Specification has the property that adding a new device requires editing and recompiling the Channel DEFINITIONS module, and thus all the modules which include it. This is in conflict with the desire not to recompile the system except for the semiannual releases (assuming new devices are to be added between releases). Thus we propose a revised Channel interface, which, it is claimed, solves the recompilation problem.

The problem with the current interface stems from its dependence on the enumerated type Channel.DeviceType, which has a value for each implementation of the interface. Adding a new implementation requires adding a new value to DeviceType, and also adding corresponding variants to the definitions of the DeviceType-tagged variant record types DeviceSelection, PhysicalRecordStatus, DeviceCommand, DeviceStatus, and "Rep" (the private record to which a Handle points). (It is worth noting that these variant records constitute a nuisance for both client and implementor. Most values crossing the interface must be wrapped in a variant record constructor, and then "unwrapped" with a one-armed WITH statement: essentially doing at runtime what Mesa is meant to do at compile time.)

A solution

Half of the operations in the current interface do not depend on DeviceType, and so are not a problem:

Delete
Suspend
Restart
GetPhysicalRecord
PutPhysicalRecord

Those which do depend on DeviceType are:

Create	(accepts DeviceSelection)
WaitPhysicalRecord	(returns PhysicalRecordStatus)
CommandDevice	(accepts DeviceCommand)
GetDeviceStatus	(returns DeviceStatus)
WaitDeviceStatusChange	(accepts and returns DeviceStatus)

A simple way to remove this dependency is to remove these operations from the Channel interface, and to add device-specific version of each of them into each specific device interface. Taking the floppy disk for example we would have:

```
FloppyDrive.Create: PROCEDURE[spindle: CARDINAL ← FloppyDrive.defaultSpindle]
    RETURNS[Channel.Handle];

FloppyDrive.defaultSpindle: CARDINAL = LAST[CARDINAL];

FloppyDrive.WaitPhysicalRecord: PROCEDURE[Channel.Handle, Channel.IOEvent]
    RETURNS[FloppyDrive.CompletionCode];

FloppyDrive.CompletionCode: TYPE = RECORD[. . .];          --as currently

FloppyDrive.CommandDevice: PROCEDURE[Channel.Handle, FloppyDrive.Command]
    RETURNS[Channel.Handle];

FloppyDrive.Command: TYPE = RECORD[. . .];   --as currently

FloppyDrive.GetDeviceStatus: PROCEDURE[Channel.Handle]
    RETURNS[FloppyDrive.Status];

FloppyDrive.Status: TYPE = MACHINE DEPENDENT RECORD[. . .];          --as currently

FloppyDrive.WaitDeviceStatusChange: PROCEDURE[Channel.Handle, FloppyDrive.Status]
    RETURNS[FloppyDrive.Status];
```

Discussion

It is interesting to note that in all cases, each variant of the records specified in the Channel interface consist of a single field whose type had already been defined in a separate device interface; the above device-dependent operations just pass that object directly. The argument that forcing all device operations to go through the Channel interface will assure uniformity from the diverse community of device interfacers seems weak in this light; providing models in the form of actual device interfaces produced here at SD/Palo Alto would seem to go a long way towards achieving useful consistency.

Implementation comments

Both the current Channel interface and the above variation presuppose a "streamlike" implementation in which every Channel.Handle points to a record which has in standard positions implementation-specific procedure descriptors for the generic operations Get, Put, etc. (Since Create doesn't take a Channel.Handle as parameter in the current interface, it would have to dispatch to an appropriate routine using a global table.) In the variation proposed above, the "DeviceType dependent" operations would be IMPORTed directly by the client from a device interface. To maintain type-checking, each of these operations accepting a Channel.Handle parameter should logically check the "device type" of the channel. This could be done with a numeric type field and type codes assigned by the Pilot maintainers, but a simpler way is available. The procedure descriptors already in the record pointed to by the Handle constitute a type code! Thus the floppy disk CommandDevice routine might begin with something like

```
IF channel.GetPhysicalRecord~=FloppyDrive.GetPhysicalRecord THEN ERROR . . .
```