JetSend: An Appliance Communication Protocol

Peter Williams
Hewlett-Packard Laboratories Bristol, England
pmw@hplb.hpl.hp.com

Abstract: Computer peripherals such as printers, scanners and digital cameras only interact with host computers by virtue of device specific ***drivers.*** *If we want such devices to interact directly with each other as information appliances, the driver architecture will only work if we make all appliances reprogrammable. The only alternative is to standardise the wire protocols of different devices, but if we do this on a case-by-case basis, the set of devices with which a given device can interact will be fixed. JetSend is a universal wire protocol for a wide class of devices which is device and device type independent. This paper explores the costs and benefits of the JetSend approach.*

1 Introduction.

The protocols used by computer peripherals are abstracted by an API (application programmer interface) for each peripheral *type*. Converting API calls into the peripheral's wire protocol is the job of the *device driver* - an extension of the peripheral in host software. The API-driver approach allows peripherals of the same type to have different wire protocols, while appearing "the same" to computer applications, as shown in Figure 1.

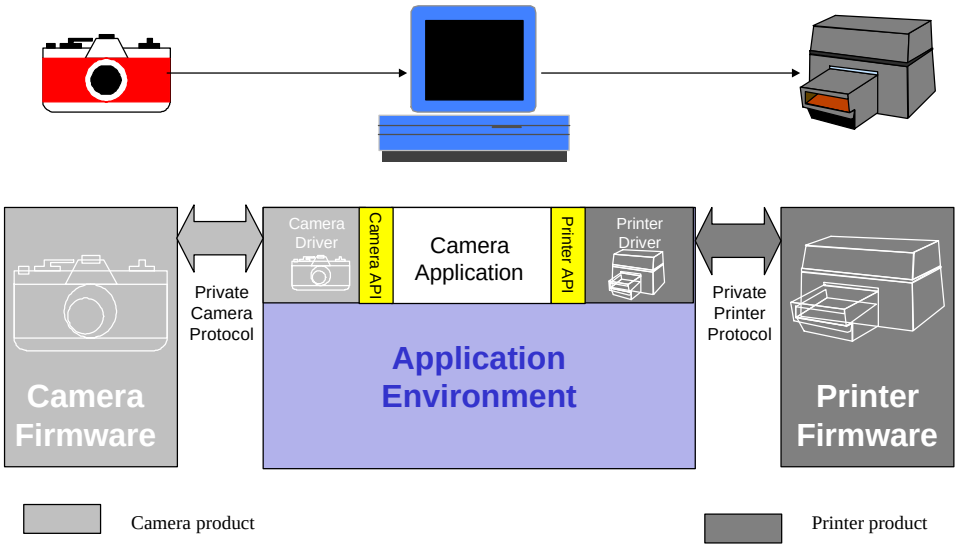

Figure 1: Peripherals, API's and Drivers

Different wire protocols make it hard to connect arbitrary pairs of peripherals unless every device can load a driver for every other, which is neither economic nor practical today. There are two ways to overcome this difficulty, as shown in figure 2:

1. Standardise both the device API's *and the software platforms*, so that a single driver for a device can be loaded into any other. Since the API-driver architecture supports private wire protocols, this will work for existing peripherals if they can be wired.
2. Standardise the wire protocols for a given type of device. This is not *necessarily* incompatible with an API-driver architecture, but devices would only need one "driver" for each other device type, and it could be pre-loaded. Standardised wire protocol result in cheaper devices, but would not support existing peripherals[1]

JetSend is a wire protocol, but it is the same for *all* types of device, making the protocol both device *and device type* independent. This means you can use the same protocol for any number of devices, as shown in figure 3. JetSend also allows new kinds of device, and new device features to be introduced without changing any *other* device. Physical appliances, such as hi-fi or kitchen appliances, also work together without needing to be adapted for each other, which is why JetSend is called an *appliance* communication protocol.

---

[1] unless *their* wire protocol is adopted as the standard for all others of the same type

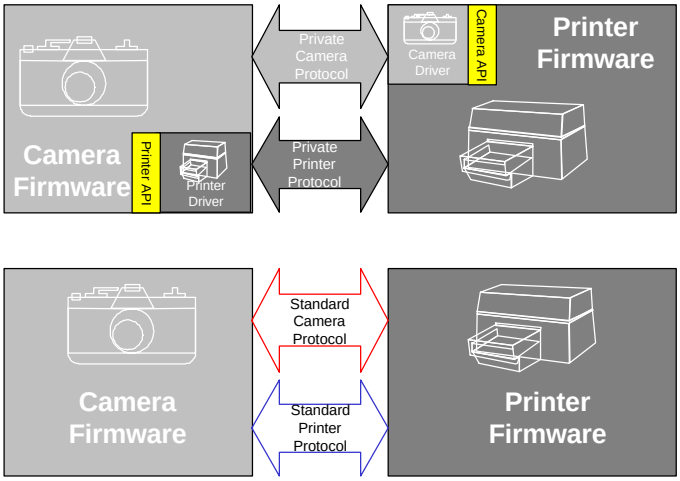

Figure 2: API+driver vs Standard Wire Protocols for Direct Connection

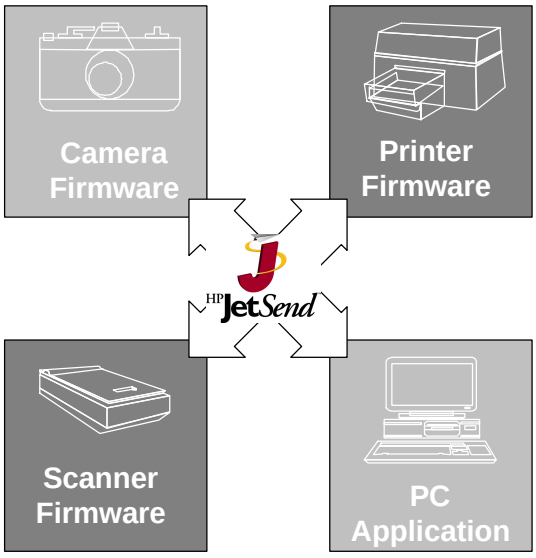

Figure 3: The JetSend Protocol

2 Standard API's, Software Platforms and Drivers.

API's and drivers make all printers look the same to Windows application, and vice versa, but the relationship is not symmetrical. Applications can be infinitely diverse, but printers can only differ along the dimensions defined by their API, and accessible peripheral *types* are limited to those for which there is an API. If applied to *direct* device communication, the API-driver model requires *all* devices to be reprogrammable both above and below the device API.

Below the API the **driver**, developed by the target device vendor, must be loaded when devices connect. Current drivers are complex, and many printers do not even have drivers for all versions of *Windows*. To be practical for device interaction, this approach requires not just a standard API but a standard *software platform* - for example, a Java virtual machine - so that the same driver can be loaded in every other device.

Above the API, the application must be developed by the device vendor. It will need to be updated to support the latest version of any target device API, or if it is to use the API for a new type of device. Device API's, and software platforms such as Java virtual machines, will inevitably be changed and extended several times over the lifetime of a device.

3 Standard Wire Protocols.

A device that cannot be reprogrammed cannot load drivers or applications, so architecturally it does not matter whether it uses a standard API, or a particular software platform. Such a device will *always* talk to

other devices in the same way, using standard wire protocols, but there are still two basic ways to standardise wire protocols:

1. By defining and parameterising the functions of a particular kind of device, which is *logically* equivalent to defining an API[2] for that device class. This results in a **function-oriented** protocol standard of the kind that device standards bodies tend to define.
2. By simply defining and parameterising the information that moves *between* devices. This results in a **data-oriented** protocol standard. Early World Wide Web and Group III Facsimile protocols were data-oriented rather than device function-oriented.

Since non-reprogrammable devices cannot change their protocols, the set of device types with which they can interact is limited to the devices which understand the protocol they use. JetSend is a data-oriented protocol which is intended for all types of device, making it both device *and device type* independent.

Traditional hi-fi "protocols" are like this. Plug an audio source into a loudspeaker and you get a sound. Plug the same source into a tape recorder and you get a recording. The signal is the same, and the "protocol" simply governs how to encode *sound*. There are no separate speaker and recorder protocols. JetSend does this for printers, scanners, cameras etc. It describes particular kinds of information, and is therefore usable by *any* device that produces, consumes, or is otherwise influenced by those kinds of information.

4 JetSend.

JetSend will clearly work for simple devices used in simple ways. A camera sending a picture to a printer could obviously do so in the same way that a scanner does, and either could interact with a TV in just the same way as they do with a printer. A PC could talk to any of these devices, and none of them, not even the PC, would need to be reprogrammed for JetSend interaction.

It is not so clear how JetSend copes with more sophisticated devices. How does a JetSend device which is not reprogrammable invoke different functions of other JetSend devices?

JetSend devices are activated by changes to their externally accessible state, by user action, or both. Their behaviour results in either a change to their externally accessible state, some real world effect, or both. The externally accessible state of a JetSend device consists of a set of *surfaces* which are the electronic analogies of the plugs, sockets, slots, bins, trays, lights and gauges found on familiar physical appliances.

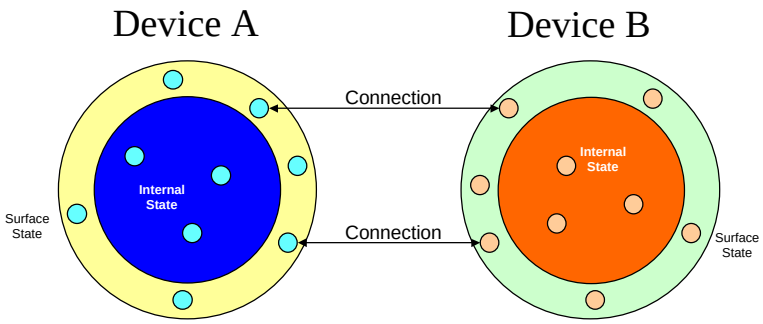

Figure 4: JetSend Surface Interaction

Figure 4 shows how JetSend devices interact by connecting their surfaces through a communication network. This style of interaction is called "surface interaction" [Williams]. Connected surfaces have identical (or rather, isomorphic) state, so the JetSend protocol consists of:

1. connecting surfaces
2. propagating changes between connected surfaces and synchronising their content
3. disconnecting surfaces

A multifunction JetSend device can therefore only be induced to perform different functions by connecting to different surfaces, or changing the same surface in different ways. And just as many device functions can be structured and parameterised, so JetSend surfaces can be structured to contain other surfaces.

As an example of a simple interaction, every JetSend printer has a default "IN" surface whose content it normally prints and then deletes. A JetSend camera, or a JetSend scanner both have a default "OUT" surface whose content normally consists of a photograph the camera has taken, or an image the scanner has

---

[2] The JINI™ proposal [SUN] uses Java remote method calls as the basis for its wire protocol. However, since JINI™ assumes a standard device API, driver, *and* universal software platform it belongs in section 2.

scanned. If the camera or scanner connects its OUT surface to the printer's IN surface, the two surfaces synchronise their content, the photograph or image is printed, and the devices disconnect.

A slightly more complex interaction occurs when a camera or scanner monitors the printer. In this case, the camera or scanner connects to the printer's "STATUS" surface and remain connected while it prints. The printer continuously changes its STATUS surface, a change which propagates to the camera or scanner, enabling it to present the printer's status to the user on its own display. In this way, a user can monitor the behaviour of a remote printer from a device which knows nothing about printers, using a wire protocol which is printer independent.

Default IN and OUT interaction and monitoring the STATUS surface are examples of JetSend device *policies* which enable simple but universal device interaction without introducing device semantics into the protocol. These simple policies are being extended [Macer] to cover more general device control, where the features and functions of remote devices can be presented to users of local devices which do not embody any knowledge of those features or functions. Abstract control surface encodings allow the user's device to determine how best to present the information to the end user, depending on the display and input transducers it has at its disposal. However, the JetSend protocol itself remains device and device independent, and does not include device-specific tokens of any sort.

In this way, JetSend can support the interaction needs of a wide range of devices:

- Scanners
- Cameras
- Displays
- Printers
- Whiteboards
- Storage devices
- TV's
- Projectors
- Desktop PC's
- Portable PC's
- PDA's
- Pagers
- Cell-phones
- Medical instruments
- Sensors
- Web servers

The *disadvantage* of JetSend is that application programs have no device API through which they can control precisely what a JetSend device does. Inasmuch as there *is* a "JetSend API" it is confined to the operation of the protocol and some standard policies. An application programmer wanting to make double-sided prints will have a couple of issues with JetSend:

1. How do you find a printer?
2. How do you tell it to print duplex?

Note that these are not issues for the *end user*. A user has many ways to find a printer - including walking up to one, knowing its network address, or being able to consult an (orthogonal) directory of printers - and JetSend will enable him to examine the printer's control interface from whatever device he has and select the double-sided option (if there is one). If the user had wanted to project his holiday photographs he could just as easily find a projector and see and select from the projection options available. But application programs are not as flexible as people - they have a very rigid view of the world, and JetSend by itself does not offer them any help.

If JetSend devices become pervasive, JetSend may need to provide programmatic access to those devices that have standard API's. This can be done by defining "machine readable" device surfaces which provide the information a conventional device API needs. This will, of course, be limited to the *standard* features and functions of the devices concerned, because an application program cannot exploit a feature or a device unknown to its programmer - even though an end user can do so easily, courtesy of JetSend.

5 Conclusion.

The JetSend protocol is a device and device *type* independent protocol *designed* for non-reprogrammable devices, and extensible to meet the interaction needs of a large set of information appliances. It is a universal, data-driven protocol, with no device specific commands, built on a simple set of surface interaction primitives. It supports and enables interaction between devices of all types, and supports end user remote control in a device independent manner. It is not specifically designed for programmatic access to device features, but can support an API+driver access model if required.

6 References.

[Macer] Peter J. Macer, Ubiquitous Remote Control of Networked Appliances using the JetSend Communication Protocol, IWNA 98, Kyoto Japan, November 1998

[SUN] http://java.sun.com/products/jini/

[Williams] Peter Williams, Surface Interaction, in: "Building Interactive Systems: Architectures and Tools" Ed. Gray and Took, Springer-Verlag 1992.