A. Title of the invention

On device device driver

B. Cross-reference to related applications

Not applicable.

C. Statement regarding federally sponsored research or development

Not applicable.

D. The names of the parties to a joint research agreement

Emile Michel Hobo

E. Incorporation-by-reference of material submitted on a compact disc or as a text file via the office electronic filing system (EFS-web)

Not applicable.

F. Statement regarding the prior disclosures by the inventor or a joint inventor

I only disclosed information on this invention to the USPTO and in daily conversation may have mentioned it, but never got to elaborate on it other than to the USPTO in the writings as filed under this patent application number.

G. Background of the invention

1. Field of the invention

This invention relates to memory models for making use of a utility within electronics and computing, focusing on device interaction or more specifically client-server interaction to facilitate making use of any kind of device that might relate as a service provider to a main computing device as a client.

2. Description of related art

Device drivers as they have existed up to now always related to a kernel as a part of its resource allocation program and had to be integrated in the client operating system to be able to make use of the service provider it was connected to, as illustrated in figure 1. For instance, in case of a printer you would have to install a device driver for that printer on the computing device that makes use of it for a user to be able to use the printer with that computing device. Different operating systems would need different device drivers and device drivers would often be upgraded to make them more efficient and take care of security issues. This would require a considerable amount of time and effort on behalf of programmers and users. Although automatic updating has reduced the effort needed to maintain and use devices, it still requires effort, knowhow, and availability of these updated device drivers to be able to use the device to its fullest potential with minimal security risk.

Other than that, even when updating device drivers, ironically the device itself would keep doing the same thing, provided that earlier device drivers didn't limit its operation of use, so as long as it works, the user typically wouldn't notice a difference in terms of the final product being delivered, just its delivery time. Also, using external drivers increases the risk of devices being used for other things than their intended use, also as access points to networks that should normally be closed. A printer should really be just that, a printer, not a router. The same goes for cameras, scanners, microwaves, general house appliances associated with the house's atmosphere and security, and any other device one might connect to ones computer.

Even with plug-and-play, when connecting devices through for instance USB, the device would only communicate its type, make, and model, but not its interaction model. This means that even with plug-and-play you were dependent on the availability of the device driver, which sometimes required additional steps when installing the device and the option to use it.

As a final observation, note the number of device drivers an operating system now has to incorporate and then imagine that only one of them were actually used, as also illustrated in figure 1. Programmers have gotten sloppy also, integrating functions in the compiled program that it never uses, which also pushes up the size of operating systems and programs alike, including device drivers which are also programs.

Now imagine a world with operating systems and even programs no bigger than the old standard of 640 kilobytes that can do everything modern operating systems and programs can do. Device drivers are program functions the way they are implemented now and if an operating system and computer program in general doesn't incorporate anything but the core functionality it uses, this cuts down on size tremendously. If you only have to design an interface once for all devices, that would even further reduce the size of the interaction code, eliminating the need for hundreds, maybe thousands of device drivers, of which most will never actually be used.

H. Brief summary of the invention

The invention is that the device driver becomes fully integrated within the device that acts as a service provider as a part of its read only memory, for the client to read it when it connects to it, so it can instantly make use of it. The device driver consists of a unique identifier, the type of device it is, the domain of the input values of the device, the range of the output values of the device, and the communication protocol used to interact with it and make use of it.

I. Brief description of the several views of the drawings

Drawings can be found on page 10.

Figure 1 illustrates the classical situation of an operating system integrating device drivers in its core functionality for resource allocation, which greatly pushes up its size, making it less efficient as shown. Here, every device driver is a program.

Figure 2 illustrates the proposed organization of storing the device driver in read only memory on the device itself. This means that a large part of the core data incorporated in the device driver like the domain and range also, now are stored separately on the device itself, greatly reducing the complexity and size of the programming of the operating system in multiple ways. An operating system only needs one interpreter rather than many after downloading the interaction specification to be interpreted in its cache. The interpreter now no longer is a part of the device driver, meaning that the driver no longer is a program, but just an interaction specification.

J. Detailed description of the invention

For a client to make use of a service provider's functions, first one needs to establish a connection. This can happen via (1) a direct physical link or (2) through a network link of one or more physical or non-physical node connections in between server and client or (3) a direct non-physical link.

When connected through a direct physical link, meaning that there are no service providers in between the client and the server and it makes use of a cable that only transfers the signal without any signal modulation or reading in between, meaning that anything linking cables turning them into one big cable shouldn't buffer but only make use of true bypass, no authentication is necessary by the human operator that uses both client and server unless specifically demanded for additional security.

In all other cases use should be made of authentication. Authentication can happen either through two-factor authentication or by setting up the server client connection through a direct connection first in such a way that they may make use of any other connection. Two factor authentication means that the user or users of server and client have to enter a code demanded by the the server into the client with this code not being communicated to the client but through a different connection or to the users directly.

Once a connection has been established in this way to guarantee the highest security available without limiting the ease of use, meaning that you still need to secure the physical location of the devices to make sure they are safe, otherwise any connection requires authentication, the client may read the server control data that allows the client to use the server without having to have a set of driver functions or programs built into its kernel other than an interpreter that interprets the server control data it downloads into its cache. This means it just downloads the device id, type, domain, range, and protocol, i.e. the proper language to speak with the device and operate it within the limitations of the device itself, which has it stored on a read only memory chip (ROM) or any of its subtypes: Programmable ROM, Erasable Programmable ROM, Electrically Erasable Programmable ROM, Flash Erasable Programmable ROM, Mask ROM, or any other type of ROM meant for ROM functionality yet to be invented.

The server control data uses a to any device to be recognized fixed construct in order for any client to make use of it, using a common XML template. The form of this template should be known and readable to all parties producing devices acting either (1) as service providers of any kind or (2) clients of any kind in order for anyone to be able to use any device with their client.

When you connect a service provider to a client and it doesn't need authentication, it triggers an interrupt. The interrupt triggers the client to read the server control data from the service provider ROM chip. In case of any other connection where authentication is required, after authentication the requirements have been met and as such it than triggers the interrupt all the same, causing the client to download the server control data.

For the server control data XML file structure, the following conventions guarantee that all devices that use these conventions can read it.

(1) The server control data file as stored on the service provider starts with `<device>` and ends with `</device>` and in between these two tags all the necessary device data are noted. Before anything else between these two tags, a unique identification needs to be specified.

(2) To do so, the file should for the identification have an opening tag `<id>` and a closing tag `</id>` with in between these two tags:

(2a) the device name between opening tag `<name>` and closing tag `</name>` and

(2b) serial number between opening tag `<ser>` and closing tag `</ser>` that together form a unique identification for a single device.

(3) After the identification for every device function it should note as an opening tag `<function>` and as a closing tag `</function>` identifying the unique and separate functions of the device, with between these two tags:

(3a) first the type of function between opening tag `<type>` and closing tag `</type>` to identify what it is this function does, followed by

(3b) its parameters within an opening tag `<parameters>` and closing tag `</parameters>` within which

(3b1) it notes the domain of the input within opening tag `<input>` and closing tag `</input>` and

(3b2) it notes the range of the output within opening tag `<output>` and closing tag `</output>` and

(3b3) for both the input domain and the output range within the respective opening and closing tag, for every variable it notes an opening tag `<var>` and closing tag `</var>` with either

(3b3a) between the variable tags the type identifier between opening tag `<type>` and closing tag `</type>` and

(3b3b) between the variable tags the name of the variable between opening tag `<name>` and closing tag `</name>` and

(3b3c) between the variable tags the input domain range or the output range between opening tag `<range>` and closing tag `</range>` with

(3b3c1) between the range tags when any range is applicable without bounds the denominator `any` or

(3b3c2) when including value ranges or specific values, recognizing all XML conventions to separate text from code, for instance by adding a slash in front of characters you mean to include as characters in the specification, like to include to include a , you type `\,` and to include a `\` you type `\\` with

(3b3c2) at least a minimum value between opening tag `<min>` and closing tag `</min>` or

(3b3c3) at least a maximum value between opening tag `<max>` and closing tag `</max>` or

(3b3c4a) at least a sequence with an opening tag `<seq>` and a closing tag `</seq>` with in between those tags a list of possible values, step ranges indicated by a low value and a high value separated by a double dot .. with every possible value and/or step range of values separated by a comma, and

(3b3c4b) between the sequence tags you may find nested all tags specified for in between the parameter tags within the same framework as proposed for all of those tags or

(3b4) when a variable has a fixed value for a specific device, instead the variable tag gets as an argument `fixed!` for the opening tag, meaning that between the opening tag `<var fixed!>` and the closing tag `</var>` one notes the constant value or sequence of values assigned to this variable for this device, meaning that the device can receive no other values than this value in terms of its operations and

(3c) every parameter definition for a device should at least contain one protocol used to interact with the device in terms of the sequence of input and output variables to be transmitted in the order to be determined in this sequence, with this sequence using the opening tag `<protocol>` and closing tag `</protocol>` with the protocol identified between those tags to adhere to the rules as applicable to everything noted between sequence tags.

In all cases, when addressing the service provider to use it, the client first notifies it what function it wishes to use, then it just sends the input variables and awaits the output response as specified in the protocol until either the client or the service provider concludes its processing in full, after which when necessary the client may address the service provider for a new task.

K. Claim or claims

1. A novel approach to register device interaction protocols and functionality by integrating the device protocols and input domain range and output range in read only memory (commonly referred to as ROM) of any kind in the device itself,

the device acting as a service provider, rather than making it a part of the kernel or any other fixed part of the device that serves as a client with the intent of making use of the service provider's functionality, meaning that no program needs to be included in the client that tells you how to make use of the service provider,

rather it only needs a program to read and interpret the protocol and operation range, input and output, of devices that can than be used with any device that makes use of this new approach.

2. A means for structuring the mechanism as described through XML that can than be installed on the ROM of any kind of the service provider for the client to download it on connecting with a secure connection, so it may make use of the server's provided functionality.

L. Abstract of the disclosure

The unique identity, type, domain, and range of interpretation as well as the protocol to be used for a client to interact with a service provider should be integrated on a ROM of any kind on the service provider to be downloaded by the client on establishing a secure connection so the client kernel can interpret it with the one and only interpreter program it needs and directly make use of the service provider without having any device drivers integrated in the client kernel itself.

M. Sequence listing

Not applicable.

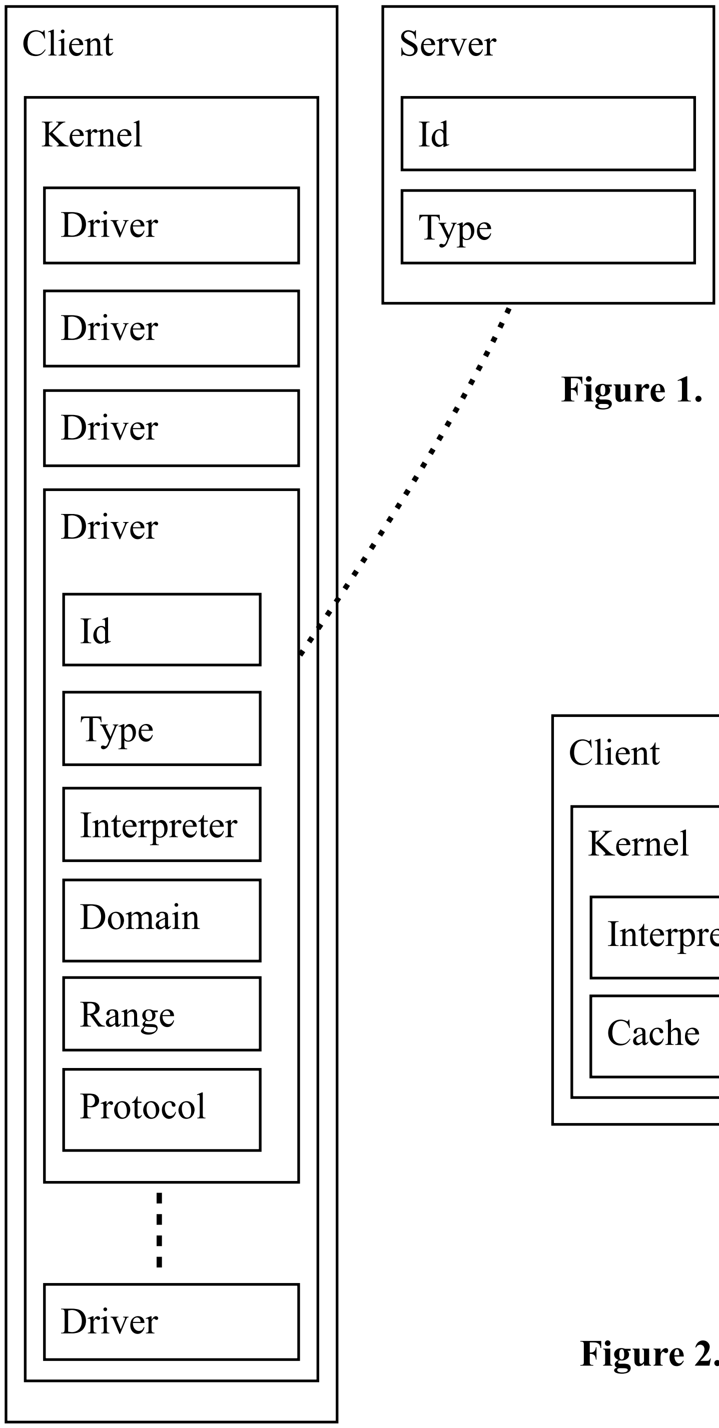

Figure 1. Classic organization implementing device drivers.

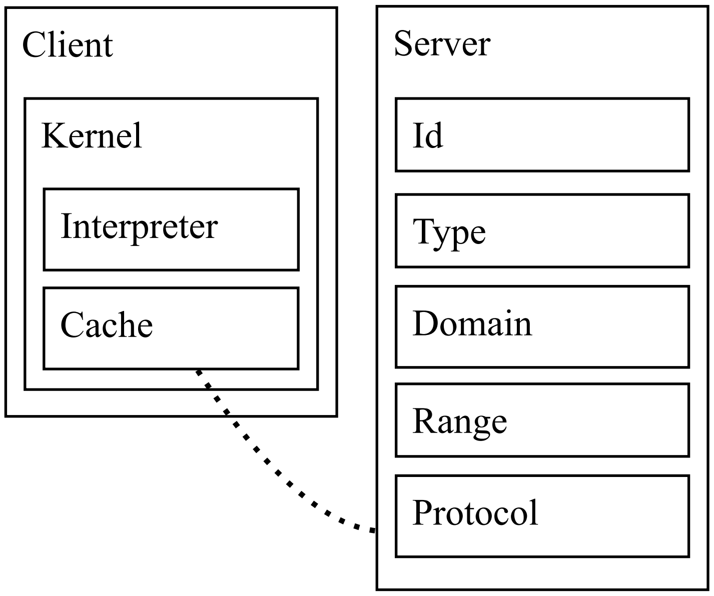

Figure 2. Proposed organization implementing device drivers.