MOTION CONTROL SYSTEMS

TECHNICAL FIELD

The present invention relates to motion control systems and, more particularly, to interface software that facilitates the creation of hardware independent motion control software.

BACKGROUND OF THE INVENTION

The purpose of a motion control device is to move an object in a desired manner. The basic components of a motion control device are a controller and a mechanical system. The mechanical system translates signals generated by the controller into movement of an object.

While the mechanical system commonly comprises a drive and an electrical motor, a number of other systems, such as hydraulic or vibrational systems, can be used to cause movement of an object based on a control signal. Additionally, it is possible for a motion control device to comprise a plurality of drives and motors to allow multi-axis control of the movement of the object.

The present invention is of particular importance in the context of a mechanical system including at least one drive and electrical motor having a rotating shaft connected in some way to the object to be moved, and that application will be described in detail herein. But the principles of the present invention are generally applicable to any mechanical system that generates movement based on a control signal. The scope of the

present invention should thus be determined based on the claims appended hereto and not the following detailed description.

In a mechanical system comprising a controller, a drive, and an electrical motor, the motor is physically connected to the object to be moved such that rotation of the motor shaft is translated into movement of the object. The drive is an electronic power amplifier adapted to provide power to a motor to rotate the motor shaft in a controlled manner. Based on control commands, the controller controls the drive in a predictable manner such that the object is moved in the desired manner.

These basic components are normally placed into a larger system to accomplish a specific task. For example, one controller may operate in conjunction with several drives and motors in a multi-axis system for moving a tool along a predetermined path relative to a workpiece.

Additionally, the basic components described above are often used in conjunction with a host computer or programmable logic controller (PLC). The host computer or PLC allows the use of a high-level programming language to generate control commands that are passed to the controller. Software running on the host computer is thus designed to simplify the task of programming the controller.

Companies that manufacture motion control devices are, traditionally, hardware oriented companies that manufacture software dedicated to the hardware that they manufacture. These

3

software products may be referred to as low level programs. Low level programs usually work directly with the motion control command language specific to a given motion control device. While such low level programs offer the programmer substantially complete control over the hardware, these programs are highly hardware dependent.

In contrast to low-level programs, high-level software programs, referred to sometimes as factory automation applications, allow a factory system designer to develop application programs that combine large numbers of input/output (I/O) devices, including motion control devices, into a complex system used to automate a factory floor environment. These factory automation applications allow any number of I/O devices to be used in a given system, as long as these devices are supported by the high-level program. Custom applications, developed by other software developers, cannot be developed to take advantage of the simple motion control functionality offered by the factory automation program.

Additionally, these programs do not allow the programmer a great degree of control over the each motion control device in the system. Each program developed with a factory automation application must run within the context of that application.

PRIOR ART

In the following discussions, a number of documents are cited that are publicly available as of the filing date of the present invention. With many of these documents, the Applicant is not

aware of exact publishing dates. The citation of these documents should thus not be considered an admission that they are prior art; the Applicant will take the steps necessary to establish whether these documents are prior art if necessary.

As mentioned above, a number of software programs currently exist for programming individual motion control devices or for aiding in the development of systems containing a number of motion control devices.

The following is a list of documents disclosing presently commercially available high-level software programs: (a) *Software Products For Industrial Automation*, iconics 1993; (b) *The complete, computer-based automation tool* (IGSS), Seven Technologies A/S; (c) *OpenBatch Product Brief*, PID, Inc.; (d) FIX Product Brochure, Intellution (1994); (e) Paragon TNT Product Brochure, Intec Controls Corp.; (f) WEB 3.0 Product Brochure, Trihedral Engineering Ltd. (1994); and (g) AIMAX-WIN Product Brochure, TA Engineering Co., Inc. The following documents disclose simulation software: (a) ExperTune PID Tuning Software, Gerry Engineering Software; and (b) XANALOG Model NL-SIM Product Brochure, XANALOG.

The following list identifies documents related to low-level programs: (a) Compumotor Digiplan 1993-94 catalog, pages 10-11; (b) Aerotech Motion Control Product Guide, pages 233-34; (c) PMAC Product Catalog, page 43; (d) PC/DSP-Series Motion Controller C Programming Guide,

pages 1-3; (e) Oregon Micro Systems Product Guide, page 17; (f) Precision Microcontrol Product Guide.

The Applicants are also aware of a software model referred to as WOSA that has been defined by Microsoft for use in the Windows programming environment. The WOSA model is discussed in the book Inside Windows 95, on pages 348-351. WOSA is also discussed in the paper entitled WOSA Backgrounder: Delivering Enterprise Services to the Windows-based Desktop. The WOSA model isolates application programmers from the complexities of programming to different service providers by providing an API layer that is independent of an underlying hardware or service and an SPI layer that is hardware independent but service dependent. The WOSA model has no relation to motion control devices.

The Applicants are also aware of the common programming practice in which drivers are provided for hardware such as printers or the like; an application program such as a word processor allows a user to select a driver associated with a given printer to allow the application program to print on that given printer.

While this approach does isolates the application programmer from the complexities of programming to each hardware configuration in existence, this approach does not provide the application programmer with the ability to control the hardware in base incremental steps. In the printer example, an application programmer will not be able to control each stepper motor in the printer using the provided printer driver;

instead, the printer driver will control a number of stepper motors in the printer in a predetermined sequence as necessary to implement a group of high level commands.

The software driver model currently used for printers and the like is thus not applicable to the development of a sequence of control commands for motion control devices.

OBJECTS OF THE INVENTION

From the foregoing, it should be clear that one primary object of the invention is to provide improved methods and devices for moving objects.

Another more specific object of the present invention is to obtain methods and apparatus for designing and deploying motion control devices in which these methods and apparatus exhibit a favorable mix of the following characteristics:

(a) allow the creation of high-level motion control programs that are hardware independent, but offer programmability of base motion operations;

(b) hide the complexities of programming for multiple hardware configurations from the high-level programmer;

(c) can easily be extended to support additional hardware configurations; and

(c) transparently supports industry standard high-level programming environments.

SUMMARY OF THE INVENTION

The present invention is, in one form, a method of moving an object comprising the steps of

7

developing a high-level motion control application program comprising a sequence of component functions that describe a desired object path, correlating these component functions with driver functions, selecting a software driver for the specific hardware configuration being controlled, generating control commands from the driver functions and the software driver associated with the hardware configuration being controlled, and controlling a motion control device based on the control data to move the object along the desired object path.

In another form, the present invention is a method of generating a sequence of control commands for controlling a motion control devices to move an object along a desired path. An application program comprising a series of component functions defines a sequence of motion steps that must be performed by the motion control device to move the object along the desired path. The component functions contain code that relates the component ~~34~~ functions to driver functions. The driver functions are associated with ~~contain~~ contain software drivers containing driver code for implementing the motion steps on a given motion control device. The control commands are generated based on the application program and the driver code associated with a given motion control device.

The use of component functions that are separate from driver functions isolates the programmer from the complexities of programming to a specific motion control device. This

arrangement also allows a given application program to be used without modification for any motion control device having a software driver associated therewith.

The driver functions are grouped into core driver functions and extended driver functions. All software drivers must support the core driver functions; the software drivers may also support one or more of the extended driver functions, although this is not required.

Where the software drivers do not support the extended driver functions, the functionality associated with the extended driver functions can normally be simulated using some combination of core driver functions. In this case, the method of the present invention comprises the steps of determining which of the extended driver functions are not supported by the software driver and, where possible, substituting a combination of core driver functions. In some cases, the functionality of an extended driver function cannot be emulated using core driver functions, and this functionality is simply unavailable to the programmer.

The use of core driver functions to emulate extended driver functions provides functionality where none would otherwise exist, but the preferred approach is to provide a software driver that supports each of the extended driver functions. When an extended driver function is supported and not emulated, the task being performed will normally be accomplished more quickly and accurately.

9

Additionally, to simplify the use of emulated extended driver functions, the method of the present invention further comprises the steps of determining which, if any, extended driver functions are not supported by the software driver for a given hardware configuration, developing a function pointer table of both unsupported extended driver functions and supported extended driver functions, and consulting the table each time an extended driver function is called to determine whether that extended driver function must be emulated. In this manner, the process of calling the sequence of core driver functions employed to emulate the unsupported extended driver functions is optimized.

As the control commands are generated as described above, they may be used to control a motion control device in real time or they may be stored in a file for later use. Preferably, the method of the present invention comprises the step of providing a number of streams containing stream code. Each stream is associated with a destination of control commands, and the stream code of a given stream dictates how the control commands are to be transferred to the destination associated with that given stream. The user is thus provided the opportunity to select one or more streams that dictate the destination of the control commands.

To help isolate the programmer from hardware specific complexities, the method of the present invention may comprise the additional administrative steps such as selecting a driver

associated with a particular motion control device and/or translating units required to define the motion control system into the particular system of units employed by a given motion control device.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a system interaction map of a motion control system constructed in accordance with, and embodying, the principles of the present invention;

FIG. 2 is a module interaction map of a motion control component of the system shown in FIG. 1;

FIG. 3 is an object interaction map of the component shown in FIG. 2;

FIGS. 4, ~~through~~ 5, 6, 7 and 8 are scenario maps of the component shown in FIG. 2;

FIG. 9 is an interface map of the component shown in FIG. 2;

FIG. 10 is a data map showing one exemplary method of accessing the data necessary to emulate extended driver functions using core driver functions;

FIG. 11 is a module interaction map of the driver portion of the system shown in FIG. 1;

FIG. 12 is an object interaction map of the driver portion shown in FIG. 11;

FIGS. 13, ~~through~~ 14, 15, 16, 17, 18, 19 and 20 are scenario maps related to the driver shown in FIG. 11;

FIG. 21 is an interface map for the driver shown in FIG. 11;

RG 4-10-97

11

PB 4-7-97

FIG. 22 is a module interaction map of the streams used by the system shown in FIG. 1;

FIG. 23 is an object interaction map of the streams shown in FIG. 22;

FIGS. 24 ~~through~~ 25, 26, 27, 28, 29, 30, 31 and 32 are scenario maps of the streams shown in FIG. 22;

FIG. 33 is an interface map of the objects comprising the stream shown in FIG. 22;

FIG. 34 is a module interaction map of the driver stub portion of the system shown in FIG. 1;

FIG. 35 is an object interaction map of the driver stub shown in FIG. 34;

FIGS. 36 ~~through~~ 37 and 38 are scenario maps of the driver stub shown in FIG. 34;

FIG. 39 is an interface map of the driver stub portion shown in FIG. 34;

FIG. 40 is a module interaction map of the driver administrator portion of the system shown in FIG. 1;

FIG. 41 is an object interaction map of the driver administrator shown in FIG. 40;

FIGS. 42 ~~through~~ 43, 44, 45, 46, 47, 48 and 49 are scenario maps relating to the driver administrator shown in FIG. 40;

FIG. 50 is an interface map of the objects that comprise the driver administrator shown in FIG. 40;

FIG. 51 is a module interaction map of the driver administrator CPL applet portion of the system shown in FIG. 1;

FIG. 52 is an object interaction map of the driver administrator CPL applet shown in FIG. 51;

12

FIGS. 53, ~~through~~ 54, 55, 56, and 57 are scenario maps related to the driver administrator CPL applet shown in FIG. 51.

13

DETAILED DESCRIPTION OF THE INVENTION

Referring now to the drawing, depicted therein at 10 in FIG. 1 is a motion control system constructed in accordance with, and embodying, the principles of the present invention. This system 10 comprises a personal computer portion 12 having a hardware bus 14, a plurality of motion control hardware controllers 16a, 16b, and 16c, and mechanical systems 18a, 18b, and 18c that interact with one or more objects (not shown) to be moved.

The personal computer portion 12 of the system 10 can be any system capable of being programmed as described herein, but, in the preferred embodiment, is a system capable of running the Microsoft Windows environment. Such a system will normally comprise a serial port in addition to the hardware bus 14 shown in FIG. 1.

The hardware bus 14 provides the physical connections necessary for the computer 12 to communicate with the hardware controllers 16. The hardware controllers 16 control the mechanical system 18 to move in a predictable manner. The mechanical system 18 comprises a motor or the like the output shaft of which is coupled to the object to be moved. The combination of the hardware controllers 16a, 16b, and 16c and the mechanical systems 18a, 18b, and 18c forms motion control devices 20a, 20b, and 20c, respectively.

The hardware bus 14, hardware controllers 16, and mechanical systems 18 are all well-known in the art and are discussed herein only to the

14

extent necessary to provide a complete understanding of the present invention.

The personal computer portion 12 contains a software system 22 that allows an application user 24 to create software applications 26 that control the motion control devices 20.

More particularly, based on data input by the user 24 and the contents of the application program 26, the software system 22 generates control commands that are transmitted by one or more streams such as those indicated at 28a, 28b, 28c, and 28d. The streams 28 transmit control commands incorporating the hardware specific command language necessary to control a given motion control device to perform in a desired manner. As will be discussed in more detail below, the streams 28 implement the communication protocol that allows the control commands to reach the appropriate motion control device 28 via an appropriate channel (i.e., PC bus, serial port).

Using the system 22, the application program 26 is developed such that it contains no code that is specific to any one of the exemplary hardware controllers 16. In the normal case, the application program 26, and thus the user 24 that created the program 26, is completely isolated from the motion control devices 20. The user 24 thus need know nothing about the hardware specific command language or communication protocol associated with each of these devices 20; it may even be possible that the command language of one or more of the hardware controllers 16 was not

15

defined at the time the application program 26 was created.

The software system 22 comprises a combination of elements that allow the application program 26 to be completely isolated from the hardware controllers 16. In the following discussion, the framework of the software system 22 will be described in terms of a method of moving an object and/or a method of generating control commands. After this general discussion, each component of the system 22 will be described in detail in a specific operating environment.

I. Method of Generating Control Commands for Controlling a Motion Control Device to Move an Object

Initially, it should be noted that, in most situations, the method described in this section will normally but not necessarily involve the labors of at least two and perhaps three separate software programmers: a software system designer; a hardware designer familiar with the intricacies of the motion control device; and a motion control system designer. The application user 24 discussed above will normally be the motion control system designer, and the roles of the software system designer and hardware designer will become apparent from the following discussion.

The software system designer develops the software system 22. The software system designer initially defines a set of motion control

operations that are used to perform motion control. The motion control operations are not specifically related to any particular motion control device hardware configuration, but are instead abstract operations that all motion control device hardware configurations must perform in order to function.

Motion control operations may either be primitive operations or non-primitive operations. Primitive operations are operations that are necessary for motion control and cannot be simulated using a combination of other motion control operations. Examples of primitive operations include GET POSITION and MOVE RELATIVE, which are necessary for motion control and cannot be emulated using other motion control operations. Non-primitive operations are motion control operations that do not meet the definition of a primitive operations. Examples of non-primitive operations include CONTOUR MOVE, which may be emulated using a combination of primitive motion control operations.

Given the set of motion control operations as defined above, the software system designer next defines a service provider interface (SPI) comprising a number of driver functions. Driver functions may be either core driver functions or extended driver functions. Core driver functions are associated with primitive operations, while extended driver functions are associated with non-primitive operations. As with motion control operations, driver functions are not related to a specific hardware configuration; basically, the

17

driver functions define parameters necessary to implement motion control operations in a generic sense, but do not attach specific values or the like to these parameters. The SPI for the exemplary software system 22 is attached hereto as Appendix A.

The software system designer next defines an application programming interface (API) comprising a set of component functions. For these component functions, the software system designer writes component code that associates at least some of the component functions with at least some of the driver functions. The relationship between component functions and driver functions need not be one to one: for example, certain component functions are provided for administrative purposes and do not have a corresponding driver function. However, most component functions will have an associated driver function. The API for the exemplary software system 22 is attached hereto as Appendix B.

The overall software model implemented by the software program 22 thus contains an API comprising component functions and an SPI comprising driver functions, with the API being related to the SPI by component code associated with the component ~~34~~ functions.

In order for the system 22 to generate the control commands, at least two more components are needed: the application program 26 and at least one software driver such as the drivers indicated at 30a, 30b, and 30c in FIG. 1.

18

The software drivers 30 are normally developed by a hardware designer and are each associated with a single motion control device. The hardware designer writes driver code that dictates how to generate control commands for controlling the motion control device associated therewith to perform the motion control operations associated with at least some of the driver functions.

In the exemplary software system 22, the software drivers 30a, 30b, and 30c are associated with the motion control devices 20a, 20b, and 20c, respectively. As a software driver exists for each of the motion control devices 20a, 20b, and 20c, these devices 20a, 20b, and 20c form a group of supported motion control devices.

A careful review of the framework of the software system 22 as described above will illustrate that, of all the components of this system 22, only the software drivers 30 are hardware dependent.

The motion control system designer, normally also the user 24, develops the application program 26. The application program 26 comprises a sequence of component functions arranged to define the motion control operations necessary to control a motion control device to move an object in a desired manner. The application program 26 is any application that uses the system 22 by programming the motion control component 34. Applications may program the system 22 either through OLE Automation or by using any of the custom OLE interfaces making up the API.

As mentioned above, the component code associates many of the component functions with the driver functions, and the driver functions define the parameters necessary to carry out the motion control operations. Thus, with appropriately ordered component functions, the application program 26 contains the logic necessary to move the object in the desired manner.

Once the application program 26 has been written and the software drivers 30 have been provided, the user 24 selects at least one motion control device from the group of supported motion control devices 20a, 20b, and 20c. Using a driver administrator module 32, the user 24 then selects the software driver associated with the selected motion control device. This driver administrator module 32 is used to install, uninstall, register, and setup each stream.

As currently implemented, the driver administrator ~~34~~ 32 allows only one software driver to be selected. In future versions of the software system 22, the driver administrator will allow the user to select one or more software drivers.

The software system 22 thus generates control commands based on the component ~~34~~ functions contained in the application program 26, the component ~~34~~ code associated with the component 34 functions, and the driver code associated with the selected software driver 28.

As the control commands are being generated as described above, they may be directly

20

transmitted to a motion control device to control this device in real time or stored in an output file for later use. The software system 22 employs the streams 28 to handle the transmission of the control commands to a desired destination thereof.

In the exemplary system 22, the destinations of the control commands may be one or more of an output file 34 and/or the controllers 16. Other possible destinations include a debug monitor or window or other custom output mechanism defined for a specific situation. The software system designer, or in some cases the hardware system designer, will write transmit stream code for each stream 28 that determines how the control commands are to be transferred to a given one of the control command destinations 16 and 34. Using the driver administrator 32, the user 24 selects one or more of the control command destinations 16 and 34, and, later when run, the system 22 transfers the control commands to the selected control command destination 16 and/or 34 based on the transmit stream code in the stream 28 associated with the selected control command destination 16 and/or 34.

Many control command destinations such as 16 and 34 are capable of transmitting data back to the system 22. Data transmitted from a control command destination back to the system 22 will be referred to as response data. The software system designer thus further writes data response stream code for each of the streams 28a, 28b, and 28c that determines how response data is transmitted

21

from the controllers 16 to the system 22. The system 22 thus processes the response data sent by the controllers 16 based on the data response stream code contained in the streams 28.

Referring again to FIG. 1, this Figure shows that the system 22 further comprises a motion control component 35 ~~34~~ and a driver stub module 36. The motion control component module 35 ~~34~~ is the portion of the software system 22 that relates the component ~~34~~ functions to the driver functions. The motion control component module 35 ~~34~~ thus contains the component ~~34~~ code that makes the association between the component ~~34~~ functions contained in the application program 26 and the driver functions.

The driver stub module 36 is not required to implement the basic software model implemented by the system 22, but provides the system 22 with significantly greater flexibility to accommodate diverse motion control hardware configurations with minimal effort.

More particularly, when the driver stub module 36 is employed, the hardware designer need not develop driver code to implement all of the driver functions; to the contrary, the hardware designer must write driver code for implementing the core driver functions but need not write driver code to implement the extended driver functions. The software system designer provides the motion control driver stub 36 with stub code that identifies the combinations of core driver functions that are employed to emulate the functionality of extended driver functions.

22

The motion control component 24 will determine for the selected software driver 30 which extended functions, if any, the selected driver 30 supports. For ~~extend~~ extended functions that are not supported, referred to herein as non-supported extended driver functions, the motion control component 35 ~~24~~ refers to the driver stub module 36 to determine the appropriate combination of core driver functions to emulate the functionality of the non-supported extended driver functions. The system 22 thus generates the control commands necessary to implement the non-supported extended driver functions using the appropriate combination of core driver functions.

The process of determining when extended driver functions need to be emulated can be optimized by providing the motion control component 35 ~~24~~ with a function pointer table that contains a pointer to each of extended functions. When building the function pointer table, the motion control component 35 ~~34~~ checks the selected driver module 30 to see if it supports each extended function. If the selected driver module 30 supports the extended function, the motion control component module 35 ~~24~~ stores a pointer to the function, implemented by the selected driver module 30, in the table location corresponding to the extended function. In the event that the selected driver module 30 does not support the extended function, the motion control component module 35 ~~34~~ stores a pointer to the extended function implementation located in the driver stub module 36. The driver stub module 36

23

implementation of the extended function contains calls to a plurality of core functions implemented by the selected driver module 30.

Therefore, the driver stub module 36 allows the motion control system hardware designer by the hardware designer to use, with minimal time and effort, a working software driver 28 that contains driver code to implement only the core functions. The software driver 28 developed to implement the core driver functions can then be improved by developing driver code to implement extended driver functions as desired.

The use of driver code specifically designed to implement extended driver functions is, in general, preferable to relying on the driver stub module 36 to emulate the extended driver functions; driver code specifically written to implement an extended driver function will almost always obtain a more optimized implementation of the driver function than the emulation of that driver function with a combination of core driver functions.

Referring again for a moment to FIG. 1, this Figure illustrates that the system 22 additionally comprises a driver administrator CPL applet 38 and a DDE server 40. The driver administration CPL applet 38 generates the user interface through which the user 24 communicates with the driver administrator module 32. The DDE server 40 provides the software interface through which the application program 26 communicates with the motion control component module 35 ~~34~~.

24

II. MOTION CONTROL COMPONENT

The motion control component 35 ^34 will now be described in further detail with reference to FIGS. 2-10. The motion control component 35 ^34 is used by every application programming the system 22 to perform motion control operations. The major set of the API is implemented by this component. When operating, the motion control component 35 ^34 interacts with the driver administrator 32, to get the current driver, and the driver 30 and driver stub 36, to carry out motion control operations. Applications, using system 22, only interact with the motion control component 35 ^34.

This section describes the design of the motion control component 35 ^34 in three main parts. First, all binary modules that affect the component 35 ^34 are described along with their interactions with the component 35 ^34. Next, the module interaction-map is drawn in more detail to show the interactions between all C++ objects used to implement the motion control component 35 ^34. Next, the object interaction-map is tested by displaying the specific interactions that take place during certain, key process that the component 34 is requested to perform.

The module interaction-map shown in FIG. 2 displays all binary modules and their interactions with the motion control component 35 ^34. As can be seen from the module interaction-map, applications only communicate with the motion control component 35 ^34 ~~(Component)~~. From this point, the component 31 ^34

25

coordinates all interactions between the driver administrator 32, driver 30, and driver stub 36 components.

Breaking the module interaction-map and adding the interactions taking place between all C++ objects used to implement the motion control component 35 ~~34~~, produces the object interaction-map shown in FIG. 3.

Each object in the diagram is described as follows. The CCmpntDisp object is the dispatch object used to dispatch exposed interface methods. During the dispatch process, all raw data is converted into the appropriate C++ form. For example, collections of data passed between OLE components is usually packaged in a raw block of memory. The CCmpntDisp object takes care of packing outgoing data and unpacking incoming data. Data packing involves converting the data between a raw and native C++ format.

The CDriverAdmin object is used to communicate directly with the driver administrator component. All OLE related details are encapsulated within this class.

The CDriverMgr object is used to control all unit mapping taking place before calling the appropriate SPI function. The CUnitMapper object is used to do the actual mapping between units.

The CUnitMapper object is used to map units between the Part Coordinate System (PCS) and the Machine Coordinate System (MCS). Both directions of unit mapping are done by this object.

The CDriver object is used to build the SPI table containing both core and extended SPI

functions. Depending on the level of driver support, the extended functions in the SPI table may point to functions implemented in either the driver stub 36 or the driver 30.

The following discussion of FIGS 4-8 describes all main scenarios, or operations, that occur on the motion control component 35 ~~34~~. Each scenario-map displays all objects involved, and the interactions that take place between them in the sequence that they occur.

As shown in FIG. 4, before an application can use the motion control component 35 ~~34~~, it must create an instance of the object, using the CoCreateInstance OLE function, and then initialize the instance calling the exposed Initialize custom interface method implemented by the component 35 ~~34~~. FIG. 4 displays the sequence of events that take place when the Initialize method is called.

During initialization, the following steps occur. First the application must create an instance of the motion control component 34 by calling the standard OLE function CoCreateInstance. Once loaded, the application must call the component 35 ~~34~~'s exposed Initialize method. When first loaded, the component 35 ~~34~~ loads any registration data previously stored. Next, the component 35 ~~34~~ directs the CCmpntDisp to initialize the system. The CCmpntDisp directs the CDriverAdmin to get the current driver(s) to use. The CDriverAdmin, first, loads the driver administrator 32 using the standard OLE CoCreateInstance function. Next, it initializes the driver administrator. Then, it queries the

27

driver administrator for the driver(s) to use and their SPI support information. Finally, the driver administrator returns the driver(s) and the support information to the component 35 ~~34~~, and releases all interfaces used from the driver administrator component 32.

Once receiving the active driver(s) 30 and their support information, the motion control component 35 ~~34~~ passes the driver(s) 30 to the CDriverMgr and directs it to initialize the system. During its initialization, the CDriverMgr initializes the CUnitMapper. Also while initializing, the CDriverMgr initializes a CDriver for each driver used. After initializing each CDriver, the support information is used to build each SPI table inside each CDriver object. When building the SPI table, all core and supported extended SPI interfaces are queried from the driver. Also, when building the SPI table, the CDriver queries all interfaces, not supported by the driver 30, from the driver stub 36.

Referring now to FIG. 5, once the motion control component 35 ~~34~~ is initialized, the application 26 may perform operations on it. There are two types of operations that may take place on the component 35 ~~34~~: Operations that use core SPI functions, and operations that use extended SPI functions. Even though the difference between the two is completely invisible to the application using the component 35 ~~34~~, the internal interactions are different between the two. The following discussion outline these differences.

28

The following interactions take place when the component 35 (~~34~~) performs an operation that uses core SPI functions only. First the application must request the operation and pass all pertinent parameters to the component 35 (~~34~~). Next, the component 35 (~~34~~) directs the CCmpntDisp to carry out the operation. The CCmpntDisp then directs the CDriverMgr to perform the operation and passes all pertinent parameters to it. Before carrying out the operation, the CDriverMgr uses the CUnitMapper to convert all units to the Machine Coordinate System (MCS). Next, the CDriverMgr directs the CDriver object to carry out the operation and passes the newly mapped parameters to it. The CDriver object uses its internal SPI table to communicate directly with the core SPI function implemented by the driver component.

FIG. 6 shows the sequence of events that occurs when the component 35 (~~34~~) is directed to carry out an operation that happens to use extended SPI not supported by the driver 30. The following steps occur when the operation is requested.

First the application must request the operation and pass all pertinent parameters to the component 35 (~~34~~). Next, the component 35 (~~34~~) directs the CCmpntDisp to carry out the operation. The CCmpntDisp then directs the CDriverMgr to perform the operation and passes all pertinent parameters to it. Before carrying out the operation, the CDriverMgr uses the CUnitMapper to convert all units to the Machine Coordinate System (MCS). Next, the CDriverMgr directs the CDriver object to carry out the operation and passes the newly

29

mapped parameters to it. The CDriver object uses its internal SPI table to communicate directly with the core SPI function implemented by the driver component.

As briefly discussed above, when using the system 22, there are several types of units and two different coordinate systems used. The process of unit mapping involves converting measurements between the Part and Machine coordinate systems. FIG. 7 illustrates this process, and the following steps occur when the operation is requested.

First the application must request the operation and pass all parameters to the component 35^34. Note, all parameters are in the PCS. Next, the component 35^34 directs the CCmpntDisp to carry out the operation. The CCmpntDisp directs the CDriverMgr to carry out the operation and passes the PCS parameters to it. The CDriverMgr takes all measurements and uses the CUnitMapper to convert them to the MCS. The newly mapped parameters are then passed to the Cdriver. The CDriver directs either the driver or the driver stub component to carry out the operation.

When the application is finished using the motion control component 35^34 it directs the component 35^34 to free all of its resources by calling its exposed Release method. This process is depicted in FIG. 8. During the clean-up process, the following steps occur.

First the application must direct the component 35^34 to release all of its resources by calling its Release method. When invoked, the

30

component 35 ~~34~~ passes the call on to the CCmpntDisp object. The CCmpntDisp object directs the CDriverMgr to release any resources it is using. The CDriverMgr directs each CDriver object to release any of its resources, then deletes the CDriver objects. First, the CDriver object releases any interfaces it is using from the driver component. Then, the CDriver object releases any interfaces it is using from the driver stub component.

FIG. 9 is an interface map related to the motion control component 35 ~~34~~. FIG. 10 is a data map showing how data relating to the whether extended driver functions need to be emulated is stored. Attached hereto as Appendix C is a document that describes the actual OLE Interfaces exposed, the definitions of the data structures used when passing data around, and the definitions of each class used internally by the motion control component 35 ~~34~~.

III. SOFTWARE DRIVERS

The driver 30 is used by both the driver administrator 32 and the component ~~34~~ 35. Its main purpose is to implement functionality that generates motion control commands for the specific hardware supported. For example, the AT6400 driver, used to control the Compumotor AT6400 motion control hardware, generates AT6400 command codes. During the initialization phase of the system 22, the driver administrator 32 communicates with each driver 30, allowing the user to add, remove, or change the configuration of the driver. When an application, using the system 22, is run, the component ~~34~~ 35 communicates with the driver 30 directing it to carry out the appropriate motion control operations.

This section describes the complete design of a generic driver 30. All drivers are designed from the base design described herein ~~in this manual~~. This section is divided into three parts. First, a module interaction-map that describes all binary modules that interact with the driver 30 is discussed. Next, the module interaction-map is drawn as an object interaction-map, where all the internals of the driver are exposed. In this map, all C++ objects, making up the driver, and their interactions are shown. Next, several scenario-maps are drawn. Each scenario-map displays the interactions taking place between the C++ objects involved during a certain process. Finally, this section describes the interfaces exposed by the

driver component, all data structures used, and the definitions of each C++ class used.

Referring now to FIG. 11, the module interaction-map displays all binary modules and their interactions with the driver 30. There are two modules that interact directly with the driver: the motion control component 35 ~~34~~, and the driver administrator 32. The driver administrator 32 queries and changes the driver settings and the component 35 ~~34~~ directs the driver to carry out motion control operations, such as moving to a certain location in the system. Shown at 42 in FIG. 11 is the standard Windows registration database, referred to herein as the registry.

Breaking the module interaction-map down into more detail by including the interactions taking place between all C++ objects used to implement the driver, produces the object interaction-map. The object interaction-map for the driver 30 is shown in FIG. 12.

Each object in the diagram is described as follows.

CDriverDisp is the dispatch object used to dispatch exposed interface methods. During the dispatch process, all raw data is converted into the appropriate C++ form. For example, collections of data passed between OLE components is usually packaged in a raw block of memory. The CDriverDisp object takes care of packing outgoing data and unpacking incoming data. Data packing involves converting the data between a raw and native C++ format.

33

The CStreamMgr object is responsible for managing the set of streams registered with the driver. Streams, may be added, removed, and enabled. Only enabled streams are sent data. The CLSID and enabled status of each stream registered, is stored in the registration database. When communicating to streams, the CStreamMgr is used to send the command string to all enabled streams.

The CCommandMgr object is used to build commands sent to the stream, and extracting responses received from the stream. The CCommandMgr is the controlling object that manages the CResponse, CCommandList, and CStream objects.

The CCommandList object stores the complete list of commands making up the motion control command language. Such commands may be stored as text resources or in a text file.

The CCommand object builds command strings that are then sent to the CStream. Each command built is a complete motion control command string.

The CResponseList object builds CResponse objects that are initialized with the parsing format for the expected response.

The CResponse object converts raw response strings, returned by the CStream, and converts them into C++ data types. For example, a response string containing position data may be converted into a set of double values.

The CStream object is used to communicate directly with the underlying stream component.

Figures 14-20 contain scenario maps that describe all main scenarios, or operations, that

occur on the driver 30. Each scenario-map displays all objects involved, and the interactions that take place between them in the sequence that they occur.

There are two types of operations that occur on the driver 30. First, the driver administrator 32 may initiate operations, such as adding streams or configuring the driver. Next, the motion control component ~~34~~ 35 may initiate operations on the driver when an application is actually running. The following discussion describes each perspective, starting with the operations directed by the Driver Administrator; all operations made on the driver by the driver administrator are discussed in the order that they may occur when using the driver.

Before a driver may be used, it must be registered in the OLE system. In order to register a driver the driver administrator first verifies that the module being registered is actually an driver 30, then it calls the DLLRegisterServer exported function to register the driver. Each module of the system 22 exports a function called DLLGetModuleType. This function is used to verify that the module is an driver 30 component. FIG. 13 displays the interactions that take place when registering a driver.

During the registration process shown in FIG. 13, the following steps occur. First, the driver administrator must load the DLL, containing the stream component, verify that the module is an driver 30. To do so, the driver administrator calls the DLLGetModuleType function, exported by

35

the driver. If the function returns a value that contains the value XMC_DRIVER_MT in the high byte, then the driver administrator proceeds and registers the driver by calling its exported function, DLLRegisterServer. When called, the implementation of the DLLRegisterServer writes all OLE 2.0 registration information to the Windows registration database.

Referring now to Figure 14, after the driver is registered, the driver administrator can load the component 35 ~~34~~ using the OLE CoCreateInstance function. During the initialization process, the driver loads all registration data and initializes both the CDriverDisp and CStreamMgr C++ objects.

During initialization, the following steps occur.

Before loading the driver component, the driver administrator must query the driver module for its CLSID. Calling the driver's exported function, DLLGetCLSID, returns the CLSID. Once it has the CLSID, the driver administrator may create an instance of the driver by calling the standard OLE function CoCreateInstance. When first loaded, the driver loads any registration data previously stored. Next, the driver directs the CDriverDisp object to initialize the system. When notified, the CDriverDisp object initializes itself and then directs the CStreamMgr to initialize itself. During its initialization, the CStreamMgr loads all stream settings from the registration database. For example, the CLSID and enabled state of all streams previously registered with the driver, are loaded.

36

After initializing the driver, the driver administrator may perform operations on it. For example, the driver administrator may request the driver to add or remove a stream. FIG. 15 displays the sequence of events occurring when the driver is requested to add a new stream. When adding a stream, the following steps occur.

First the driver administrator directs the stream to add a new stream and passes CLSID of the stream, to be added, to the driver. The driver then passes the CLSID to the CDriverDisp object and directs it to add the stream. The CDriverDisp object passes the information on to the CStreamMgr and directs it to add the stream. In the final step, the CStreamMgr assumes that the module is a valid stream component 28 and adds the CLSID to the drivers set of information in the registration database.

Another operation requested of the driver, after initialization, is that of querying it for its current settings. Before displaying information about the driver, like the name of the hardware it supports, the driver administrator must query the driver for the information. For example, FIG. 16 displays the process of querying the driver for an enumeration of the streams registered with it. When querying the driver for information, the following steps occur.

First the driver administrator, calls the interface method used to query the driver's stream enumeration. Next, the driver directs the CDriverDisp to create the stream enumeration. The CDriverDisp object then directs the CStreamMgr to

37

prepare the stream enumeration. The CStreamMgr checks the registration database and makes sure its internal state is in sync with the data stored in the registry. Next, it sets a lock that will cause all stream management operations, such as adding or removing streams, to fail. The CStreamMgr prepares the list of streams and loads them into memory using the CStream object. The CStream object loads the stream component using the OLE CoCreateInstance API.

After the driver administrator is done using the driver, it must release the driver by calling its exposed Release method. Calling this method, directs the driver to release all resources used. FIG. 17 displays the process of releasing the driver component. During the clean-up process, the following steps occur.

First the driver administrator must direct the driver component to clean itself up by calling its Release method. When invoked, the driver component passes the call on to the CDriverDisp object. The CDriverDisp object then directs the CStreamMgr to save all data. The CStreamMgr saves all data, including the state of each stream, in the registration database. Finally, the driver saves all internal data in the registration database.

After a driver is successfully installed into the system 22 and configured using the driver administrator, it is ready for use by the motion control component 35 ~~34~~. The component 35 ~~34~~ uses the driver 30 when performing motion control operations requested from the application using

38

the component 35 ~~34~~. The following discussion describes the component 35 ~~34~~ directed operations that can take place on the driver.

Before using the driver, it must be initialized by the component 35 ~~34~~. This operation is different from the driver initialization taking place on the driver when used by the driver administrator because the system must be prepared for sending and receiving commands. In order to prepare for the data communication, the stream must be initialized and then opened. FIG. 18 describes the initialization process. The following steps occur during the initialization process.

First the component 35 ~~34~~ must direct the driver to initialize itself. This is usually a two step process. In the first step, the component 35 ~~34~~ creates and instance of the driver using the standard OLE CoCreateInstance function. Next, the Initialize method, exposed by the driver, is called to prepare the driver for data transmissions. When the Initialize method is called, the driver first loads any internal data stored in the registration database 42. Next, the driver directs the CDriverDisp to initialize the internal system. The CDriverDisp then directs the CStreamMgr to initialize the streams. Next, the CStreamMgr loads all data from the registration database, including the set of all CLSID's and enabled status' for all streams registered with the driver. Then the CStreamMgr loads each enabled stream by creating a new CStream object for each enabled stream. When creating each

39

CStream object, the CLSID for the underlying stream is passed to the CStream object. When each CStream object is created and attached to a stream component it loads the component 35 ~~34~~ by calling the standard OLE CoCreateInstance function. Once the CStreamMgr is done, the CDriverDisp directs the CCommandMgr to initialize itself. During its initialization process, the CCommandMgr initializes and loads the CCommandList. Also, when the CCommandMgr is initializing, it loads the CResponseList corresponding to the CCommandList.

Once the system is initialized, the motion control component 35 ~~34~~ can direct the driver to carry out certain command operations. Command operations are standard motion control operations such as moving to a specific location in the system, or querying the system for the current position. FIG. 19 describes the process of commanding the driver to carry out a certain operation. When commanding the driver to perform a certain operation the following steps occur.

First, the component 35 ~~34~~ directs the driver to perform the operation, such as moving to a position or querying the system for the current position. Next, the driver directs the CDriverDisp object to perform the operation. The CDriverDisp object then directs the CCommandMgr to build the appropriate command. Any parameters related to the command are passed to the CCommandMgr. For example, when directing the driver to move to a certain position, the position information is passed to the CCommandMgr. Next, the CCommandMgr requests the CResponseList to

40

create a CResponse object. The CResponseList looks up the response format and uses it to create a new CResponse object that is returned to the CCommandMgr. Then, the CCommandMgr directs the CCommandList to create the command. Any parameters related to the command are passed to the CCommandList. The CCommandList creates a new CCommand object, looks up the raw command string, and passes it and the command parameters to the CCommand object who then builds the command string.

The CCommandMgr, then passes the CCommand object, returned by the CCommandList, and the previously created CResponse object to the CStreamMgr object. The CStreamMgr object is directed to process the objects. The CStreamMgr passes the CCommand and CResponse objects to all enabled CStream objects. The CStream object queries the CCommand object for the full command string in raw text form. The raw text command is passed to the stream component. Next, the CStream object waits for the response, then reads the raw text response into a buffer. The raw text response is then passed to the CResponse object. Next the CRETONNE object is returned to the CStreamMgr, who returns it to the CCommandMgr, who returns it to the CDriverDisp object. Eventually the CResponse returns to the CDriverDisp object, who then directs the CResponse to convert the response into a generic C++ type. The generic type is returned to the motion control component 35 ~~34~~.

41

Once the component ~~34~~ 35 is finished using the driver, the driver must be released by calling its Release method. Releasing the driver frees all resources used by the driver. FIG. 20 describes the process of releasing the driver. The following steps occur when cleaning up and freeing all resources used by the driver.

First, the component ~~34~~ 35 must call the driver's Release method. When called, the driver directs the CDriverDisp object to release any resources used. The CDriverDisp then directs the CStreamMgr to free any resources used. The CStreamMgr then frees all active CStream objects. Each CStream object releases all stream component interfaces used. Next the CDriverDisp directs the CCommandMgr to free all of its resources. During its clean-up, the CCommandMgr frees the CCommandList object. To complete its clean-up, the CCommandMgr frees the CResponseList object.

Attached hereto as Appendix D is a document that describes the actual OLE Interfaces exposed, the definitions of the data structures used when passing data around, and the definitions of each class used internally by the driver.

IV. STREAMS

This section describes the stream component 28 used as the data transport layer between the driver 30 component and the destination output location such as the motion control device 20 and/or the output file 34. For example, when using motion control hardware that is connected to the PC Bus, the driver 30 Component will communicate with the PC Bus stream component 28.

The design of a stream component 28 will be discussed in three parts. First, a Module Interaction-Map describes the modules that are involved, with respect to the stream, and how they interact with one another. Next, the Object Interaction-Map breaks the Module Interaction-Map down into a more detailed view that not only displays the interactions occurring between modules, but also the interactions taking place between the C++ objects within the stream component 28. Then, the Object Interaction-Map is "tested" by running it through several Scenario-Maps. Each Scenario-Map displays the object interactions taking place during a certain operation.

The Module Interaction-Map shown in FIG. 22 displays all modules that interact with the stream component 28. Interactions begin from two different perspectives. First, the driver administrator 32 interacts with the stream component 28 when installing, removing, and configuring the stream. Next, when used, each driver 30 interacts with the stream while sending

43

and retrieving data to and from the destination. For example, when a driver writes data to a text file stream, the stream takes care of writing the data out to the file. Or, if the driver reads data from a PC Bus stream, the stream does the actual read from the hardware and passes the data back to the driver.

Drivers only communicate with streams that have been specifically connected to the driver. Once connected, the stream is used to communicate with the destination object, like the PC Bus, serial I/O connection, text file, or debug monitor.

The stream component 28 shown in FIG. 22 is the object that operates as the data transport layer for each driver. Each stream has a different target that defines the type of the stream. The following are the current stream targets.

PC Bus/WinNT - This Windows NT stream uses a Windows NT .SYS device driver to communicate directly with the motion control hardware connected to the PC Bus.

PC Bus/Win95 - This Windows 95 stream uses a Windows 95 VxD to communicate directly with the motion control hardware connected to the PC Bus.

PC Bus/Win 3.1 - This Windows 3.1 stream communicates directly with the motion control hardware connected to the PC Bus.

Serial - This stream uses the COMM API to communicate with the motion control hardware connected to the serial port.

Text File - This stream is write-only and sends all data to a text file.

Debug Monitor - This stream is write only and sends all data to the debug monitor.

Custom - This is a custom stream that sends data to an unknown location.

Similar to the Module Interaction-Map, the Object Interaction-Map displays interactions between modules. In addition, this map, shows all interactions taking place between each C++ object within the stream component 28. FIG. 23 is the Object Interaction-Map for the stream component 28.

Each object in the diagram is described as follows. The CStreamDisp object is the dispatch object used to dispatch exposed interface methods. During the dispatch process, all raw data is converted into the appropriate C++ form. For example, collections of data passed between OLE components are usually packaged in a raw block of memory. The CStreamDisp object takes care of packing outgoing data and unpacking incoming data. Data packing involves converting the data between a raw and native C++ format.

The CRegistryMgr object takes care of managing all data stored in the registration database. Since many streams of the same type may exist at the same time, each stream is assigned a handle. The handle assigned, is used by the stream to look up the location it uses to load and store data in the registration database, much as an library index is used to locate a library book.

All input and output is funnelled through the CIOMgr manager. Management of input and output operations consists of buffering data and controlling primitives used to transport data to and from the target location.

The CIOHAL object is the input/output hardware abstraction layer. With in this object lay all hardware dependent code such as calls to inp and outp. Each different type of stream contains a different implementation of this object.

Scenario-Maps are specialized Object Interaction-Maps that display how each module and the objects inside the stream component interact with one another during the operation described by the map. The Scenario-Maps in FIGS. 24-32 are broken into two different categories; those that are initiated by the driver administrator 32, and those that are initiated by the driver 30.

Operations directed by the driver administrator are usually related to initializing, uninitializing, and configuring the stream. The following sections describe all operations, directed by the driver administrator, that take place on the stream.

Before a stream component can be used by anyone, it must be registered in the Windows registration database. Registration is a standard OLE 2.0 operation required in order to use any OLE 2.0 component, such as the stream component. FIG. 24 describes this process. During the registration process, the following steps occur.

First, the driver administrator must load the DLL, containing the stream component, verify that the module is an stream component 28. To do so, the driver administrator calls the DLLGetModuleType function, exported by the stream. If the high byte in the return value contains the value XMC_STREAM_MT, then the driver administrator proceeds and registers the stream by calling its exported function, DLLRegisterServer. When called, the implementation of the DLLRegisterServer writes all OLE 2.0 registration information to the Windows registration database.

After the stream component is successfully registered, it is ready for initialization. During initialization, the stream component not only initializes itself, but also initializes any device drivers used by registering the driver with the operating system. For example, the Windows NT stream component registers the Windows NT .SYS driver with Windows NT and starts the service. FIG. 25 describes this process. During initialization, the following steps occur.

First the driver administrator must direct the stream to initialize itself. When making this call, the name and location of the driver used, and the handle of the stream are passed into the method as arguments. Once directed to initialize itself, the stream component calls the CStreamDisp and directs it to initialize the system. The CStreamDisp object then directs the CRegistryMgr to load all pertinent data for the stream using the handle passed to it. The CRegistryMgr loads all data from the registration database. After

all information is loaded from the registry, the CStreamDisp directs the CIOMgr to register the appropriate driver with the operating system. The CIOMgr directs the CIOHAL to register the driver, if appropriate. If running in Windows NT, the CIOHAL registers the .SYS driver with the Windows NT operating system and starts the driver. If running in Windows 95, the VxD integrity is verified with a quick, dynamic, load and unload.

After initializing the stream component, it may be queried for its current settings or directed to set new settings. Since both operations are very similar, only changing settings will be described. Stream settings include data such as: port addresses, IRQ levels, file names, etc. Any data needed to communicate with the output/input target are included in the stream settings. FIG. 26 describes the process of changing the streams settings. During the setup process, the following steps occur.

First the driver administrator directs the stream to use the data passed to change its internal data. Once directed, the stream component passes the interface method invocation to the CStreamDisp object. The CStreamDisp object then directs the CRegistryMgr to store the new settings. The CRegistryMgr stores the new values in the registration database.

When the driver administrator is done using a stream component, it must clean up the resources used. FIG. 27 describes this process. During the clean-up process, the following steps occur. First the driver administrator must direct the

48

stream component to clean itself up by calling its Release method. When invoked, the stream component passes the call on to the CStreamDisp object. The CStreamDisp object then directs the CRegistryMgr to save all data. All persistent data is saved to the registration database by the CRegistryMgr.

Driver directed operations occur when each driver 30 uses the stream component 28 connected to it. Remember, each stream component is used as the data transport layer. Each driver uses the stream to transfer the motion control command data, it generates, to the output target. Streams are also used to transfer data back to the driver when read operations occur. Only certain streams are readable.

Before the driver can perform operations on the stream, the stream must be initialized. Initialization occurs in two steps. First the OLE stream component must be loaded, and then once it is, the stream must be explicitly initialized. FIG. 28 describes the second portion of the initialization process. The following steps occur during the initialization process.

First the driver must invoke the Initialize methods exported by one of the stream interfaces. When calling Initialize, the driver passes to the stream, the stream handle. Next, the stream passes the directive on to the CStreamDisp object for dispatching. The CStreamDisp object first directs the CRegistryMgr to load all settings stored in the location defined by the stream handle. The CRegistryMgr reads in the data stored

in the registry at the handle. After the data is loaded, the CStreamDisp, directs the CIOMgr to initialize itself. As part of its initialization, the CIOMgr initializes the CIOHAL object that it is using.

Once a stream has been initialized, it must be opened. Opening a stream places the stream in a state where it can pass data between the driver and the target. FIG. 29 describes the process of opening a stream. When opening a stream, the following steps occur.

First the driver directs the stream to open itself, by calling the Open exposed interface method. Once directed, the stream passes the call on to the CStreamDisp object. Next, the CStreamDisp object directs the CIOMgr to open the stream. At this time, the CIOMgr prepares any buffers that will later be used when transferring data through the stream. After the buffers are ready, the CIOMgr directs the CIOHAL object to interact with the target and open it. CIOHAL directly communicates with the target or with a device driver and opens the stream. When operating with hardware streams, the device driver, or Serial IO directly communicates with the hardware and prepares it for operation.

After opening a stream, it is ready to perform data transport operations. There are two main data transport operations available: Reading data, and writing data. FIG. 30 describes the process of writing data to the stream. When writing to the stream, the following steps occur. First the driver directs the stream to write data

to the target and passes the data to the stream. Next, the stream passes the data to the CStreamDisp object. The CStreamDisp object passes the block of data to the CIOMgr and directs it to write it to the target. The CIOMgr object either passes the complete block of data to the CIOHAL object, or stores the block in an internal buffer and then passes pieces of the buffer to the CIOHAL object until the complete buffer is sent. The CIOHAL object takes the data passed to it and either sends it directly to the target, passes it to a device driver, or calls COMM API to send the data to the Serial IO port. The device driver or COMM API sends the data directly to the hardware controlled.

Certain streams, like the PC Bus and Serial IO streams, return data after write operations occur on them. The data returned may be specific to a previous request for data, or status describing the success or failure of the previous write operation. FIG. 31 describes the process of reading data from the stream. It should be noted that not all streams are readable. Currently, the only readable streams are the PC Bus and Serial streams. During the operation of reading data from the target, the following steps occur.

First the driver directs the stream to read data from the target. The stream passes the call on to the CStreamDisp object. The CStreamDisp object directs the CIOMgr to perform the read. Depending on how the stream is implemented, the CIOMgr may either make one call or multiple calls to the CIOHAL object. If multiple calls are made,

51

all data read is stored in CIOMgr internal buffers. The CIOHAL object either directly communicates to the hardware, uses the COMM API, or a device driver to read the data. If a device driver or the COMM API are used, they directly communicate with the hardware to read the data.

Once the driver is done using the stream, it must direct the stream to clean-up all resources used. To do so, the driver calls the standard Release method. FIG. 32 displays the sequence of events taking place after the Release method is called. The following steps occur when cleaning up and freeing all resources used by the stream.

First the driver must call the stream's Release method. Next, the stream directs the CStreamDisp object to release all resources used. The CStreamDisp object then directs the CIOMgr to free any resources used in buffers, etc. Next, the CIOMgr directs the CIOHAL to free any resources used. During its clean-up and depending on the type of stream, the CIOHAL will delete text files used, close the debug monitor, shut-down the hardware, or direct any device drivers to shut-down the hardware. If device drivers or the COMM API are used, they direct the hardware to shut-down.

FIG. 33 depicts an interface map for the stream 28. Attached hereto in Appendix E is a document that describes the actual OLE Interfaces exposed, the definitions of the data structures used when passing data around, and the definitions of each class used internally by the stream.

52

V. DRIVER STUB MODULE

The driver stub module 36 is used to fill in the extended SPI functions that the driver 30 is unable to support or implement. By simulating the extended SPI, applications are able to use a larger set of motion control functionality than would be available if the application directly programmed the motion control hardware. In order to implement the extended SPI, the driver stub uses software algorithms that call core SPI interface methods implemented by the driver 30. During the initialization of the driver stub, the driver 30 to use is registered with the driver stub.

This section describes all aspects of the driver stub 36 in three basic parts. The first part of this section describes all binary modules affecting the driver stub. Next, a more detailed view, that includes all C++ objects used inside the driver stub, is described. Then several processes that take place on the driver stub are described.

The module interaction-map displays all binary modules and their interactions with the driver stub 36. As can be seen from FIG. 34, the driver stub is used by the component 35 (~~34~~). More or less, the driver stub acts as a helper to the component 35 (~~34~~) by filling in all extended SPI functionality possible.

By taking the module interaction-map in FIG. 34 and displaying all interactions taking place with all C++ objects implementing the driver stub,

we produce what is called the object interaction-map. FIG. 35 is the object interaction-map for the driver stub 36 component.

Each object in the diagram is described as follows.

The CDriverStubDisp object is the dispatch object used to dispatch exposed interface methods. During the dispatch process, all raw data is converted into the appropriate C++ form. For example, collections of data passed between OLE components is usually packaged in a raw block of memory. The CDriverStubDisp object takes care of packing outgoing data and unpacking incoming data. Data packing involves converting the data between a raw and native C++ format.

The CSPIMgr object is responsible for managing all SPI issues such as managing the CSimpleDriver by directing it to connect to the appropriate SPI core interfaces exposed by the driver.

The CSimpleDriver object is used to directly communicate with the driver implementing the SPI core interfaces. The CSimpleDriver only communicates with the core SPI interfaces implemented by the driver.

The following discussion describes all main scenarios, or operations, that occur on the driver stub 36. Each scenario-map displays all objects involved, and the interactions that take place between them in the sequence that they occur. All

operations on the driver stub originate from the motion control component 35 ~~34~~. In addition to the motion control component 35 ~~34~~, the XMC Setup Component interacts with the driver stub when installing the system 22. It should be noted that all scenarios below assume that the driver stub 36 has already been registered in the OLE system. Registering this component is the responsibility of the setup application and setup component.

This discussion describes all operations made on the driver stub by the motion control component 35 ~~34~~. Each section is discussed in the order that they may occur when using the driver.

As shown in FIG. 36, before using the driver stub 36, the motion control component 35 ~~34~~ must initialize it by creating an instance of the driver stub, and then initializing the instance created. Calling the standard OLE function CoCreateInstance completes the first step. After an instance is created, the component 35 ~~34~~ must call the driver stub exposed Initialize interface method. During initialization, the following steps occur.

The component 35 creates an instance of the driver stub by calling the standard OLE function CoCreateInstance. Once loaded, the CLSID of the driver to use is passed to the driver stub when calling its Initialize exposed interface method. When first loaded, the driver loads any registration data previously stored. Next, the component 35 ~~34~~ passes the CLSID, of the driver to use, to the CDriverStubDisp object and directs it to initialize the system. The CDriverStubDisp

55

object then directs the CSPIMgr to initialize itself and passes the driver CLSID to it. The CSPIMgr passes the CLSID to the CSimpleDriver and directs it to only query the core SPI interfaces exposed by the driver. The CSimpleDriver loads an instance of the driver then queries all core interfaces exposed by the driver.

Once the driver stub is initialized, it is ready to perform operations such as performing extended SPI functions. FIG. 37 describes the steps that occur when the component 35 ~~34~~ directs the driver stub to perform an extended SPI operation. The following steps occur when the operation is requested.

First the component 35 ~~34~~ must request the operation and pass all pertinent parameters to the driver stub. Next, the driver stub directs the CDriverStubDisp to handle the operation. The CDriverStubDisp then directs the CSPIMgr to perform the SPI extended function and passes the appropriate XMC_EXT_SPI identifier as a parameter. The CSPIMgr calls the appropriate function corresponding to the XMC_EXT_SPI identifier. The function simulates the extended SPI function and calls the CSimpleDriver for core operations. When directed, the CSimpleDriver performs SPI core functions by directly calling the exposed interfaces implemented by the driver.

When the motion control component 35 ~~34~~ is finished using the driver stub 36, it must release it by calling the exposed Release method. Calling the Release method causes the driver stub to free all the resources it uses. FIG. 38 displays this

sequence of events. During the clean-up process, the following steps occur.

a First the component 35 ~~34~~ must direct the driver stub to release all of its resources by calling its Release method. When invoked, the driver component passes the call on to the CDriverStubDisp object. The CDriverStubDisp object then directs the CSPIMgr to release any resources that it was using. The CSPIMgr releases all resources including the CSimpleDriver object used. When freed, the CSimpleDriver releases any interfaces used from the driver.

FIG. 39 is an interface map of the driver stub module 36. Attached hereto as Appendix F is a document that describes the actual OLE Interfaces exposed, the definitions of the data structures used when passing data around, and the definitions of each class used internally by the driver.

57

VI. DRIVER ADMINISTRATOR MODULE

The driver administrator 32 is used from two different perspectives. When the driver administrator Control Panel Applet 38 is used to configure the system, the applet directs the driver administrator 32 to carry out the operations. The applet 38 simply provides the user-interface, and the component 35 ~~34~~ does the real work of managing drivers and streams used with the system 22. Using the driver administrator component with the control panel applet is the first perspective on using the component 35 ~~34~~.

In the second perspective, the motion control component 35 ~~34~~ uses the driver administrator component to query for the current set of enabled the driver 30. It should be noted that, currently, only single driver operation is allowed. Clearly, the system 22 may support multiple drivers that are virtualized. For example, if two, four axis, drivers are installed, applications using the system could act as though they were using an eight axis system.

This section describes the driver administrator 32 in three main parts. First, all modules interacting with the driver administrator component are described along with their interactions. Next, the module interaction-map is expanded to display all interactions taking place between the C++ objects used to implement the driver administrator 32 Component. This description is called the object interaction-map. Then, the object interaction-map is tested by

58

running it through several scenarios, or scenario-maps. Each scenario-map displays the events and the order in which they occur in a certain process taking place on the driver administrator component.

The module interaction-map shown in FIG. 40 displays all binary modules and their interactions with the driver administrator 32 Component. Both the driver administrator CPL 38 and the motion control component 35 ~~34~~ are the main modules that interact with the driver administrator 32 Component.

The driver administrator CPL module 38 provides the user-interface that allows the user to add, configure, and remove drivers and streams in the system 22. The driver administrator 32 handles all driver and stream management. Even though the control panel applet provides the user-interface, this module 32 does the actual management work.

In addition, the driver administrator is used by the component 35 ~~34~~ to access the current driver(s) to use when carrying out motion control operations. For example, if the AT6400 driver is selected as the current driver when the component 35 ~~34~~ queries the driver administrator, the driver administrator returns the CLSID of the AT6400 driver.

Taking the driver administrator 32, displayed in the module interaction-map, and displaying all interactions occurring between the C++ objects used to implement the administrator 32 ~~34~~, produces the object interaction-map therefor. The object

59

interaction-map for the driver administrator 32 is shown in FIG. 41.

Each object in the diagram is described as follows.

The CDriverAdminDisp object is the dispatch object used to dispatch exposed interface methods. During the dispatch process, all raw data is converted into the appropriate C++ form. For example, collections of data passed between OLE components is usually packaged in a raw block of memory. The CDriverAdminDisp object takes care of packing outgoing data and unpacking incoming data. Data packing involves converting the data between a raw and native C++ format.

The CDriverInfoMap object is used to build the information used by the driver administrator CPL 38 when displaying information about each driver or stream.

The CModuleMgr object is responsible for managing all stream and driver modules in the system. A list of all drivers registered are stored persistently in the registration database by the CModuleMgr. Each time a driver or stream is accessed the CModuleMgr is used to get the module.

The CSimpleDriver object is used to directly communicate with the driver component. All OLE specific details are encapsulated within this object.

The CSimpleStream object is used to directly communicate with the stream component. All OLE specific details are encapsulated within this object.

60

FIGS. 42-49 describe all main scenarios, or operations, that occur on the driver administrator 32. Each scenario-map displays all objects involved, and the interactions that take place between them in the sequence that they occur.

Referring now to FIG. 42, before using the driver administrator component, it must be initialized. FIG. 42 describes the process of initializing the driver administrator component from either the driver administrator control panel applet or the motion control component. During initialization, the following steps occur.

First, either the control panel applet or the motion control component must create an instance of the driver administrator component by calling the standard OLE function CoCreateInstance. Next, the exposed Initialize interface method must be called. When the Initialize method is called, the driver administrator component directs the CDriverAdminDisp to initialize the system. Next, the CDriverAdminDisp directs the CModuleMgr to initialize itself and any modules that it is managing. The CModuleMgr, first, loads all information from the registration database. Then for each driver registered, the CModuleMgr creates an instance of the driver by calling the standard OLE function CoCreateInstance. Next, the CModuleMgr calls each drivers Initialize method, passing to the method the CLSID of the driver component to attach. The CSimpleDriver attaches to the driver component by calling the standard OLE function CoCreateInstance.

The driver administrator 32 can register both drivers and streams. Registering drivers is very direct, since the driver administrator manages the drivers registered in the system. Registering streams, on the other hand, is more complex, since each stream must be registered with a driver and the driver manages the streams registered with it, not the driver administrator. The following discussion describes the process of registering both drivers and streams.

Registering a driver entails verifying that the module is actually a driver, verifying that the driver can be loaded, and storing the driver information in a persistent location. FIG. 43 describes this process. When registering a driver, the following steps occur.

First, the driver administrator CPL passes the name of the driver and directs the driver administrator component to register it. Next, the driver administrator component passes the driver name to the CDriverAdminDisp and directs it to register the module. The CDriverAdminDisp directs the CModuleMgr to register the new driver. The CModuleMgr creates a new CSimpleDriver and requests it to register the driver. First the CSimpleDriver verifies that the driver is valid by calling its DLLGetModuleType exported function. If the function returns XMC_DRIVER_MT the CSimpleDriver then calls the driver's exported function DLLRegisterServer to register the module in the OLE system. Next the CLSID is queried from the module by calling its exported DLLGetCLSID function. The CLSID returned is then used to load

62

the driver by calling the standard OLE function CoCreateInstance. If the CSimpleDriver is successful, the CModuleMgr stores the driver CLSID in the registration database.

Registering a stream is similar to registering a driver, but a little more complex, since each stream must be registered with a specific driver. FIG. 44 displays the process of registering a stream. When registering a stream, the following steps occur.

First, the driver administrator CPL passes the CLSID of the driver and the filename of the stream to register with the driver, to the driver administrator component. The driver administrator component directs the CDriverAdminDisp to register the stream. The CDriverAdminDisp object directs the CModuleMgr to register the stream and passes the CLSID of the driver and the name of the stream along to it. First, the CModuleMgr verifies that the CLSID of the driver one of the registered drivers. If it is not, the driver is registered as discussed above.

Next, the CModuleMgr creates a new CSimpleStream object and directs it to verify and load the stream component. The CSimpleStream first verifies that the module is actually an stream component 28 by calling its exported DLLGetModuleType function. If the function returns XMC_STREAM_MT, the CSimpleStream continues and registers the stream component by calling its DLLRegisterServer exported function. Finally, the CSimpleStream object queries the new module for its CLSID by calling the module's exported

DLLGetCLSID function. The new CLSID is used, by the CSimpleStream, to load the stream component using the standard OLE function CoCreateInstance. If the CSimpleStream succeeds, the CLSID of the stream is passed along to the CSimpleDriver who is directed to register the stream. The CSimpleDriver passes the CLSID to the driver component and directs it to register the stream.

The following discussion describes setting information in either a driver or stream. When the user edits information in the driver administrator control panel applet 38, the applet 38 directs the driver administrator 32 to edit the settings for the stream or driver being edited. The following discussion describes how this configuration process works.

Editing the settings of a driver takes place when the user changes the driver settings displayed in the driver administrator CPL. Changing these settings causes the process described in FIG. 45 to occur within the driver administrator component. The following steps occur when setting the driver configuration.

When driver settings are changed in the CPL 38, the driver administrator CPL directs the driver administrator component to make the appropriate changes to the driver corresponding to the driver handle. A XMC_DRIVER_INFO structure is passed to the component 35 ~~34~~, describing the new values for the driver. The driver administrator component takes the XMC_DRIVER_INFO structure and the handle to the driver and passes the information to the CDriverAdminDisp object,

directing it to change the settings in the driver. The CDriverAdminDisp object directs the CModuleMgr to edit the driver corresponding to the driver handle. The CModuleMgr locates the CSimpleDriver with the handle and directs it to change its settings to those stored in the XMC_DRIVER_INFO structure. The CSimpleDriver passes the XMC_DRIVER_INFO structure to the driver component and directs it to change its settings.

As shown in FIG. 46, when the user edits stream settings in the driver administrator CPL 38, the following steps occur.

After the user changes settings for the stream in the CPL, the driver administrator CPL directs the driver administrator component to change the stream's settings and passes a handle to the driver containing the stream, a handle to the stream, and a XMC_STREAM_INFO structure describing the new values. The driver administrator component directs the CDriverAdminDisp object to change the streams settings. The CDriverAdminDisp object directs the CModuleMgr to change the settings of the stream corresponding to the handle.

First, the CModuleMgr locates the driver corresponding to the driver handle. Next, it requests the CSimpleDriver to change the settings for the stream corresponding to the stream handle. The CSimpleDriver searches for the stream corresponding to the stream handle and directs it to change its settings to those stored in the XMC_STREAM_INFO structure. The CSimpleStream directly communicates with the stream component

65

and directs it to change its settings to those in the XMC_STREAM_INFO structure.

There are two different types of information that may be queried from the driver administrator 32: the enumeration of all drivers registered, and the driver information map. The motion control component 35 ~~34~~ uses the driver enumeration when selecting the set of drivers to use and control during motion control operations. The driver information map, on the other hand, is used by the driver administrator CPL 38 to update the user-interface display describing all drivers and streams registered in the system. The following discussion describes the process of querying for both the driver enumeration and the driver information map. Querying for the driver enumeration occurs during the initialization of the motion control component 35 ~~34~~. When initializing, the component 35 ~~34~~ must know what drivers to use when performing motion control operations. The driver administrator 32 Component is used for that very purpose. Querying the driver enumeration just returns a pointer to the IXMC_EnumDriver interface exposed by the driver administrator 32 Component. FIG. 47 displays the events that occur when using the interface to get each driver in the enumeration. Using the interface causes, the following steps occur.

First, the motion control component 35 ~~34~~ queries the driver administrator 32 Component for the next driver. Next, the driver administrator 32 Component directs the CDriverAdminDisp to get the next driver supported. The CDriverAdminDisp

directs the CModuleMgr to get the next driver. The CModuleMgr then directs the CSimpleDriver to either return the CLSID or a pointer to the IUnknown interface for the driver, depending on the parameters of the enumeration. If the CSimpleDriver is requested to return a pointer to the IUnknown interface, the interface is queried from the driver component.

Another set of information that may be queried from the driver administrator 32 consists of the driver information map. This data is used by the driver administrator CPL 38 when displaying information describing the drivers and streams registered in the system. As shown in FIG. 48, when querying the system for the driver interface map, the following steps occur.

First, the driver administrator CPL 38 queries the driver administrator 32 Component for the current driver information map. When queried, the driver administrator component directs the CDriverAdminDisp to create and load a CDriverInfoMap class. The CDriverAdminDisp creates the CDriverInfoMap. Next, the CDriverAdminDisp passes the CDriverInfoMap to the CModuleMgr and directs it to load the information map. The CModuleMgr queries each driver registered for its internal information. Each CSimpleDriver communicates directly with the driver component and queries it for all pertinent driver information. Next, the CModuleMgr queries each driver for a list of all streams registered with the driver. Using the stream enumeration, each CSimpleDriver creates an array of

CSimpleStream objects and returns the array to the CModuleMgr. For each CSimpleStream object in each array, the CModuleMgr queries for all pertinent stream information. Each CSimpleStream communicates directly with the stream component and queries it for all information describing the stream.

After the driver administrator CPL 38 or the motion control component 35 ~~34~~ are finished using the driver administrator 32, they must release the component 35 ~~34~~ to free any resources it was using. FIG. 49 describes this process. When cleaning up after a call to the Release method, the following steps occur.

First, either the driver administrator CPL 38 or the motion control component 35 ~~34~~ must direct the driver administrator 32 ~~Component~~ to release itself by calling its Release method. Next, the driver administrator component directs the CDriverAdminDisp object to free all resources used in the system. The CDriverAdminDisp then directs the CModuleMgr to free any resources that it is using. First, the CModuleMgr traces through all CSimpleDriver objects, querying each for their CLSID and enabled state. Next, each CSimpleDriver is freed. Each CSimpleDriver object freed, frees all arrays of CSimpleStream objects registered with it. When freed, each CSimpleStream object releases all interfaces that it was using from the stream component. In its final clean-up, each CSimpleDriver releases all interfaces that it was using from the driver component. All CLSID and

68

enabled state information is stored persistently in the registration database.

FIG. 50 depicts an interface map for the driver administrator 32. Also, attached hereto as Appendix G is a document that describes the actual OLE Interfaces exposed, the definitions of the data structures used when passing data around, and the definitions of each class used internally by the driver administrator 32 component.

VII. DRIVER ADMINISTRATOR CPL APPLET

This document describes the design of the driver administrator control panel applet (CPL) 38 ~~38 (CPL)~~ that is used by the user to add, configure, and remove both drivers 30 and stream components 28 later used by the component 35 ~~34~~ when directed to carry out motion control operations. With regard to design, there are three main types of "views" used to look at how the control panel applet works.

First, a module interaction map shown in FIG. displays all main executable and user-interactable items, or modules, that the CPL uses and interacts with. For example, when a dialog is displayed by the CPL executable, both the dialog and the CPL modules are considered to interact with one another. Technically, the dialog is not a module since it is a figment displayed on the screen, but none the less, module interaction maps classify them as such since they are key destination points for user-input.

Second, an object interaction map shown in FIG. 52 displays all main objects making up the modules described in the module interaction map. Objects consist of the actual instances of C++ classes defining each object. All interactions between the objects are drawn out in this interaction map.

Finally, FIGS. 53-57 display a set of scenario maps are drawn out using the object interaction map as a basis. Scenario interaction-maps describe the interactions taking place during

a specific operation. Initialization, Adding a driver to the system, and Viewing the support offered by a driver, are all examples of a scenario interaction-map.

The design goals for the driver administrator 32 are the following:

1. User-Interface separation - Implement all user-interface elements used to control the driver administrator 32 Component.

2. Upgradable to OCX Client - Eventually each driver and stream may implement all UI elements with an OCX that then passes all input to the corresponding driver or stream. The driver administrator CPL 38 must be designed in a way that is easy to upgrade to become an OCX client.

3. Provide Stream Independence - drivers 30 should not be required to use streams 28 in order to operate. The design of the driver administrator 32 must make amends to ensure that it is not dependent on stream component 28 operations to operate.

4. Use Windows 95 UI - When ever possible, Windows 95 UI elements should be used. For example, TreeViews, ImageLists, Button Bars, Tab Dialogs and any other UI elements should be put to use to ensure a Windows 95 look-and-feel.

The following discussion describes the module interaction map for the control panel applet 38. A module is defined as either an executable binary, an external data file, or a main user-

interface element used when interacting with the user. FIG. 51 is a drawing of all modules that interact with each other when running the driver administrator control panel applet.

The driver administrator CPL 38 is a control panel applet. And, a control panel applet is a special DLL that exports several functions allowing the Windows Control Panel to communicate with the applet.

The Driver Administrator Dialog is the main dialog that appears when selecting the control panel applet icon from the Windows Control Panel.

The Browse Dialog is used to query the user for a filename. For example when adding a new stream or driver, the driver administrator uses this dialog to ask the user for the location of the new driver or stream to add.

The View Support Dialog displays the support provided by the selected driver 30. Each driver may support a different set of extended functionality. This dialog shows the user exactly how much support is provided by each driver allowing them to determine which functions within their application may not operate when using the driver.

Unlike the Module Interaction-Map described above, the Object Interaction-Map shown in FIG. 52 describes how the actual instances of C++ objects interact with one another within each module.

Other than showing that each dialog is managed by the object, whose name is displayed in the dialog, the main difference from the module

72

IA-map are both the CComCPL and CDriverAdmin C++ objects. Both objects are described below.

As the description of each dialog class is fairly straight forward and very similar to the dialog description above they will not be described in this section. This section will describe all other C++ objects.

The CComCPL is a C++ object that is generated by the COMBuilder application from a template. It is used to handle all Windows messages sent from the Control Panel Application.

The CDriverAdmin object is used to drive, control, and manage the use of the driver administrator 32 Component. For example, all OLE 2.0 interface management and data translation is handled by this object. Data translation involves translating data from a standard C++ format to a raw format that is handled easily with the OLE 2.0 data transfer mechanisms.

Scenario Interaction-Maps are almost identical to object interaction-maps but they only display the objects and interactions taking part in a specific operation. Also, each interaction is numbered by the sequence in which they occur while the operation is running. The following discussion describes several key operations that occur while running the driver administrator CPL 38 Applet.

Initialization occurs when the user first runs the CPL Applet. During this process all other objects are initialized and several modules are loaded. There are two steps that take place during the initialization process: First the

application is initialized, and second the dialog is initialized with values queried from the driver administrator 32 ~~Component~~. The following sections describe each.

Initializing the application, which is shown in FIG. 53, occurs when the application is first run and the main dialog has not yet been displayed. When initializing the application, the following steps occur.

Through a Windows message, Windows notifies the CComCPL object that the Control Panel Applet has just been loaded. CComCPL then loads the CDriverAdminDialog and tells it to do any dialog prepping before going modal. Next, CDriverAdminDialog loads any settings stored in the Registration Database. For example, the current window position and active tab may be stored in the database. CDriverAdminDialog then Loads the CDriverAdmin class and directs it to initialize itself. During initialization, CDriverAdminDialog creates an instance of the driver administrator 32 and queries all interfaces that will be used.

Once the application is initialized, the default settings to be displayed in the dialog must be set. These values are set when the dialog is initialized, just before displaying it. FIG. 54 describes this process. During the process of initializing the dialog, the following steps occur.

During the dialog preparation that occurs before the DoModal call, CDriverAdminDialog queries the CDriverAdmin object for the driver

74

enumeration to be used when setting initial values to be displayed in the dialog box. CDriverAdmin uses the driver administrator 32 ~~Component~~ to query for the driver information map, which is then passed back to the CDriverAdminDialog. Once receiving the driver information map, the CDriverAdminDialog uses the information to update all user-interface items related to either drivers or streams.

Adding a driver to the system 22 can be broken down into two steps. First, the module name must be added to the system. Next, the driver administrator 32 main dialog must update itself to reflect the new driver just added.

Adding a driver occurs when the user presses the "Add..." button on the driver administrator 32's main dialog. FIG. 55 describes this process. When adding a new driver, the following steps occur.

When adding a driver, first the user must press the "Add..." button. After pressing the button, CDriverAdminDialog opens up the common open file dialog. The user must enter in the filename of the driver to add and close the dialog. CDriverAdminDialog then passes the filename to the CDriverAdmin object and calls the RegisterDriver method passing in the name of the module to register as a driver. CDriverAdmin then passes the driver filename to the driver administrator 32 ~~Component~~ and directs it to register the driver in the system 22.

75

The process of updating the main dialog is identical to the process of initializing the dialog discussed above.

Similar to the process of adding a new driver, removing a driver involves both removing the driver from the system and then updating the main dialog. Pressing the "Remove" button removes a driver from the XMC software system. FIG. 56 describes this process. The following steps occur when removing a driver.

To remove a driver, the user must first select the "Remove" button. After pressing the button, the selected driver or parent driver to the selected stream will be removed. CDriverAdminDialog passes the XMC_HDRIVER of the driver to the CDriverAdmin and directs it to remove the driver by calling its UnRegister method. CDriverAdmin passes the XMC_HDRIVER to the driver administrator 32 ~~Component~~ and directs it to UnRegister the driver.

The process of updating the main dialog is identical to the process of initializing the dialog discussed above.

Viewing Support involves viewing the level of support implemented by the selected driver. FIG. 57 describes the process of providing this information to the user via the View Support Dialog. The following steps occur when viewing the support provided by the driver.

First the user must select the "View Support" button on the driver administrator main dialog. When selected, CDriverAdminDialog queries CDriverAdmin for the driver support information.

CDriverAdmin passes the query on to the driver administrator 32 component who actually fills out the information. Once the queried information is returned, the CDriverAdminDialog passes it on to CViewSupportDialog. CViewSupportDialog initializes itself using the driver support information.

Attached hereto as Appendix H is a document that describes the actual OLE Interfaces exposed, the definitions of the data structures used when passing data around, and the definitions of each class used internally by the driver administrator 32.

VIII. DRIVER ADMINISTRATOR CPL APPLET

This section contains a description of the driver administrator control panel applet 38. When using the driver administrator 32 to configure the motion control system, there are two main items that the user will work with: drivers and streams. Each driver 30 generates the hardware specific, control codes that are then sent to the selected stream component 28. Streams facilitate the data transport layer between the driver and the control-code destination.

Depending on the current hardware setup, different streams may be used. For example, if the hardware is connected to the PC Bus, a PC Bus stream will be used to communicate to it. On the other hand, if the hardware is connected through a serial cable to a serial I/O Port, the serial stream will be used. Finally, all hardware configurations may use the file stream. When using the file stream, all control-codes are sent to the specified file that can be downloaded to the hardware at a later time.

This section describes both drivers and streams, and how each is configured. This section initially describes the driver items and all property pages used to edit them. This section also contains a description of the streams and their property pages. Finally, this section describes the about box containing details on the Software.

The main purpose of each driver is to generate the hardware-specific control-codes

directing the hardware to carry out specific motion control actions. For example, such actions may include querying the hardware for the current position or directing the hardware to move to a predetermined location in the system. The following discussion describes the property pages used to configure each driver.

There are two types of properties affecting each driver. First, a set of defaults may be set that are used by the motion control component 34 as recommended values. The scaling and units used are several example default values. In addition to setting default values, if the driver supports more advanced configuration, pressing the Advanced... button will display a dialog box used to set the driver configuration. For example, if a driver does not support streams, the advanced configuration dialog, provided by the driver, will allow the user to set the I/O Port and IRQ settings.

The properties affecting drivers 30 are as follows.

Scaling - Setting the scaling property affects the default scaling used on all axes within the motion control system. The range for scaling values is (0.0, 1.0]. Default setting may be overridden when programming XMC by using the IXMC_StaticState interface.

Units - Setting the units property affects all coordinates used when programming the system 22.

The unit descriptions are as follows:

MM_ENGLISH - Inches are used as the base unit for all coordinates

MM_METRIC - Millimeters are used as the base unit for all coordinates.

MM_NATIVE - The native coordinates defined by the hardware system are used. Coordinates used to program XMC are mapped 1:1 to the hardware coordinates.

Advanced... - Pressing this button will display a dialog used to edit any advanced properties for the driver that may be edited by the user.

In addition to allowing the user to set properties, each driver property page displays the full names of both the hardware supported and the hardware vendor who makes the hardware.

The buttons along the bottom of the windows work with the selected driver or stream. The following discussion describes each button and what it does.

Pressing the Make Default button selects the current driver to be the default. If a stream is selected, its parent driver becomes the default driver. The default driver is later used by the motion control component 35 ~~34~~

Selecting the Add... button, displays the Add Module dialog. This dialog is used to add new drivers and streams to the system 22. Once selected, the new driver or stream will be displayed in the Driver tree view. When adding a stream, the stream is added under the currently selected driver. To enable the stream, you must

80

select the enable check box located in the streams property page.

Selecting the Remove button, removes the current driver or stream selected. If a driver is removed all of its streams are also removed.

Selecting the View Support... button displays a dialog used to view the level of XMC support implemented by the driver. For example, all API interfaces and subsequent methods are displayed. If a lack of implementation within the driver prohibits an API interface from operating, the driver stub 36 is used. If the lack of implementation within the driver 30 cannot be replaced by operations within the driver stub 36, the interface or method is disabled.

The following are descriptions of each graphic found in the XMC Support View Dialog.

D - This graphic means that the interface or method is implemented by the driver 30.

S - This graphic means that the interface or method is implemented within the driver stub 36.

X - This graphic means that the interface or method is disabled because of a lack of implementation within the driver 30.

Like the properties page, a debug page is also provided to set all debugging settings for the driver. Each driver may specify that all API calls used to control the driver are logged. The logging settings only affect the current driver selected. The Output field allows you to select the output stream where all debug information is

sent. When Streams is enabled, debug information is sent to the specified text file. When Debug Monitor is enabled, debug information is sent to the debug monitor if it is running. Using Enable to enable a stream turns it on causing all debug information generated to be sent to the stream. More than one stream may be enabled at one time.

Stream Settings are available for each debug stream supported. Text File allows the name of the text file may be set. The Debug Monitor can only be enabled and disabled.

A stream is the transport layer used by the driver to pass data to the destination location. The destination location may be the actual motion control hardware or even a text file. Usually the control language used by a hardware vendor is supported by several different flavors of their motion control hardware. For example, some vendors have both PC Bus based and Serial I/O based motion control hardware that understand the same control language. In such a case, the same driver would be used for each hardware setup but it would communicate with different streams depending on the specific hardware setup. Graphically, each stream is listed below each driver that uses the stream.

This section describes the streams supported by the system 22 and how they are configured.

The PC Bus stream sends all data directly to a PC Bus based motion control hardware system by writing to the specified I/O Ports and IRQ's defined by the hardware. This section describes

both the properties and debug settings available for the PC Bus Stream.

Stream properties only affect the currently selected stream. The user is required to select certain settings, such as the I/O Port and IRQ. Without setting these values, the PC Bus Stream will not be able to communicate with the hardware. The properties affecting PC Bus Streams are described below.

The I/O Port is the base port used to communicate with the motion control hardware that the stream is to send data to.

The IRQ is the interrupt request level used by the hardware.

Pressing the Advanced... button will display a dialog allowing the user to edit more advanced stream options. For example, if the stream supports a Port I/O map that the user can edit, the port map would be displayed in this dialog. This button is only enabled for streams supporting advanced features that the user may edit.

When debugging an application program it may be useful to see what codes are actually sent to the hardware. The Debug Settings page for streams allows the user to enable and disable both the Cmd and Bit Streams. The Cmd Stream is used to log all command-codes sent to the hardware. If this level of detail does not provide ~~you with~~ enough information, the Bit Stream may be used. When enabled, the Bit Stream logs all values sent through each hardware port. All values read from and written to each port used by the hardware are logged. Note, when enabled, both streams may

significantly slow down the application programming the motion control system.

Serial RS-232 Streams are used to send data from the driver to motion control hardware connected to the computer through the serial I/O port. Both property and debug settings only affect the selected Serial RS-232 Stream. The following discussion describes the available settings in each in detail.

All Serial RS-232 property settings must be set by the user for they let the stream know what I/O port and communication protocol to use when communicating with the hardware. The properties affecting Serial RS-232 Streams are as described below.

The Port is the serial port that the hardware is connected to. COM1 - COM4 are valid ports that can be used.

The Baud Rate is the speed of data transmission supported by the hardware.

When Hardware is selected a more efficient, but less compatible, communication protocol is used to communicate to the hardware. If errors occur when this protocol is selected, use the XON/XOFF communication protocol.

When the XON/XOFF communication protocol is selected a simple and more compatible communication protocol is used.

Debug settings for the Serial RS-232 Stream are very similar to those supported by the PC Bus Stream. Serial RS-232 Streams only support command logging through the Cmd Stream and do not support bit logging.

84

The Text File Stream is used to build control-code programs for later use. Using this stream facilitates running the XMC software in code-generation-mode. No motion control actions take place when running in this mode. Instead, control-code programs may be built and stored to file. Later, after programs are built and saved, they may be downloaded to the motion control hardware and run. The following discussion describes the property and debug settings for the Text File Stream.

The main property set, when configuring a Text File Stream, is the actual name and location of the file to use. Once set, the stream is ready for use.

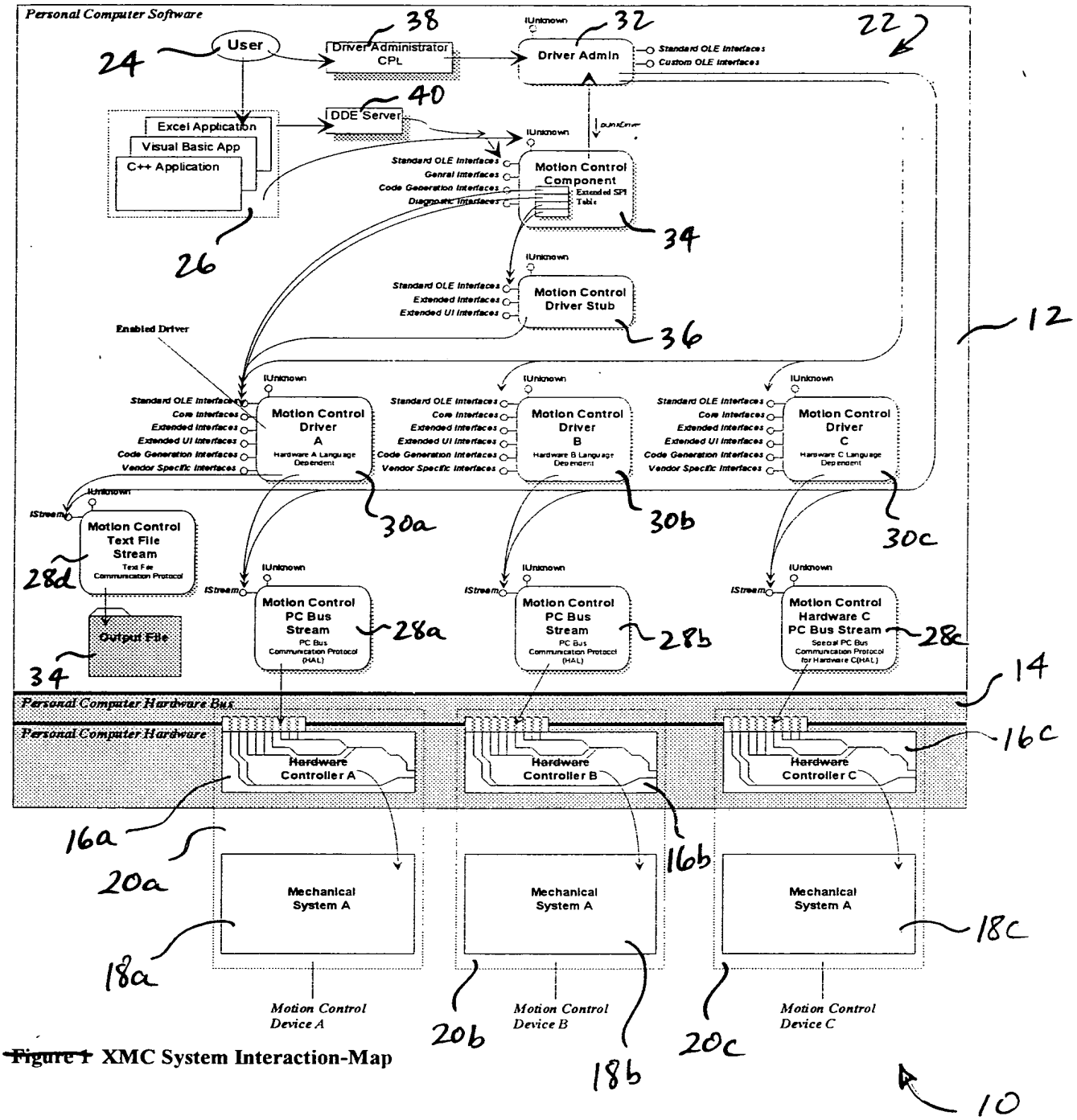
The following properties may be configured for the Text File Stream:

Filename is the filename and location of the file used to store all control-codes generated by the driver 30 selected. Pressing the Browse... button displays a dialog allowing you to graphically select the location and filename to use.

No debug settings are available for the Text File Stream.

It should be clear from the foregoing that the present invention may be embodied in other specific forms without departing from the essential characteristics thereof. The present embodiments are therefore to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by the appended claims rather than by the foregoing

description; all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced therein.



~~Figure 1~~ XMC System Interaction-Map

FIG. 1

(57)

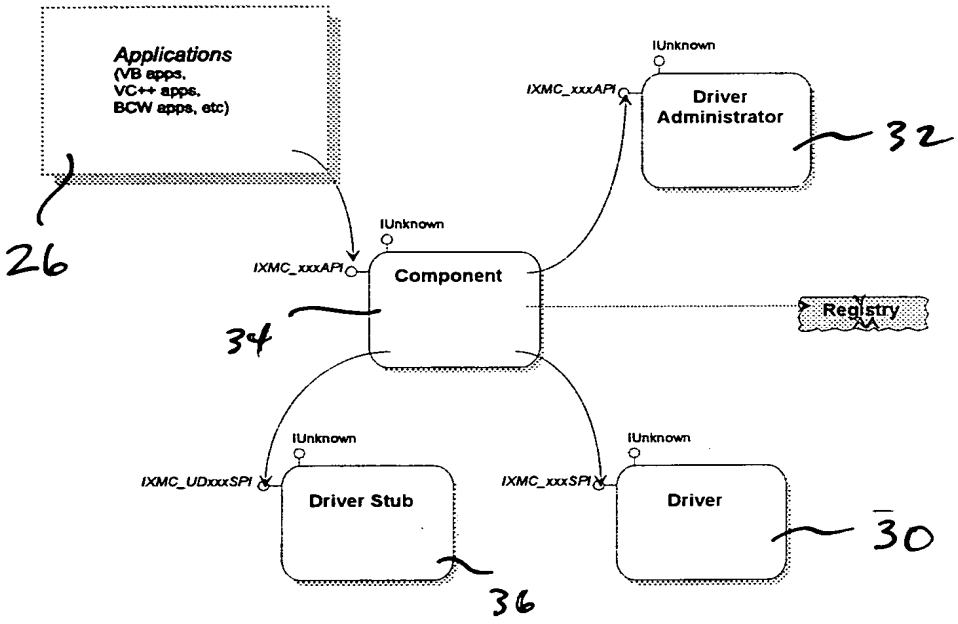
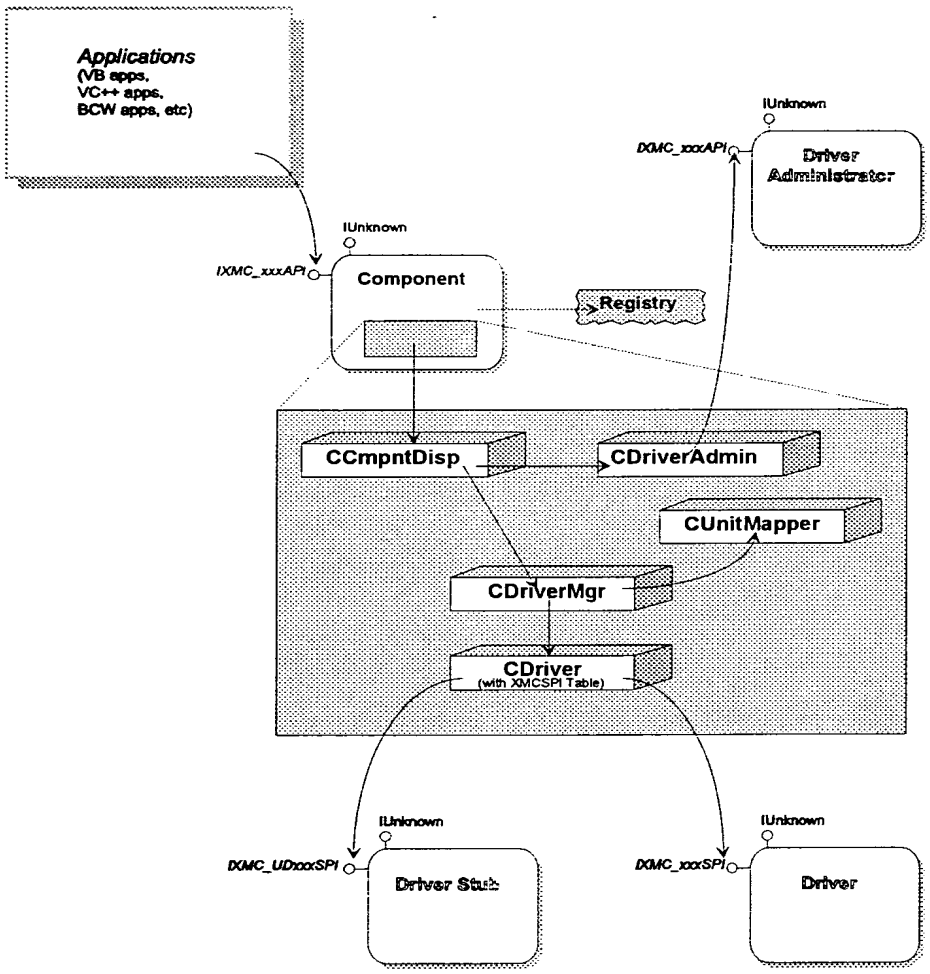


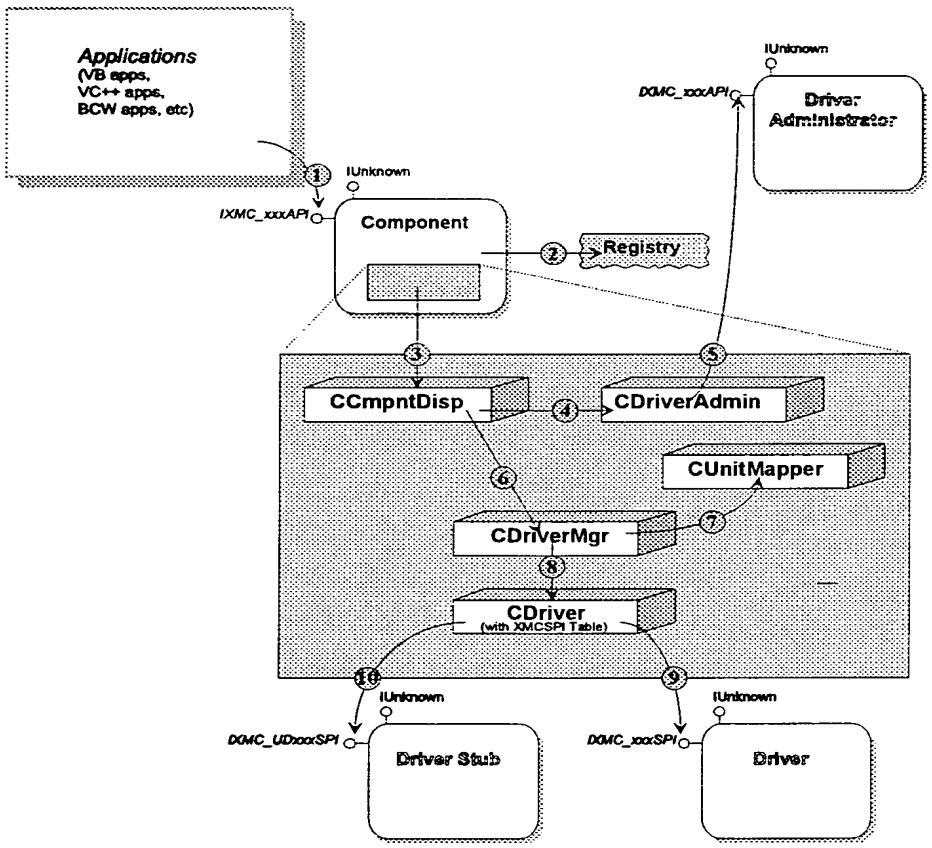
Figure 1 *Module Interaction-Map.*

FIG. 2



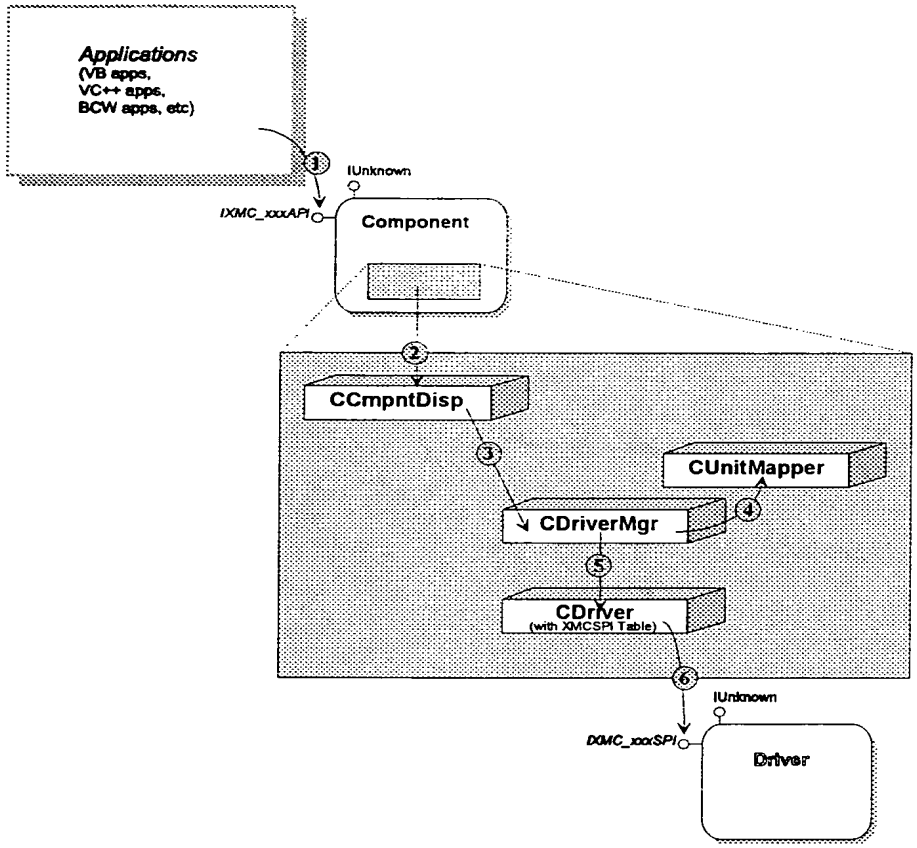
~~Figure 2~~ *Object Interaction-Map.*

FIG. 3



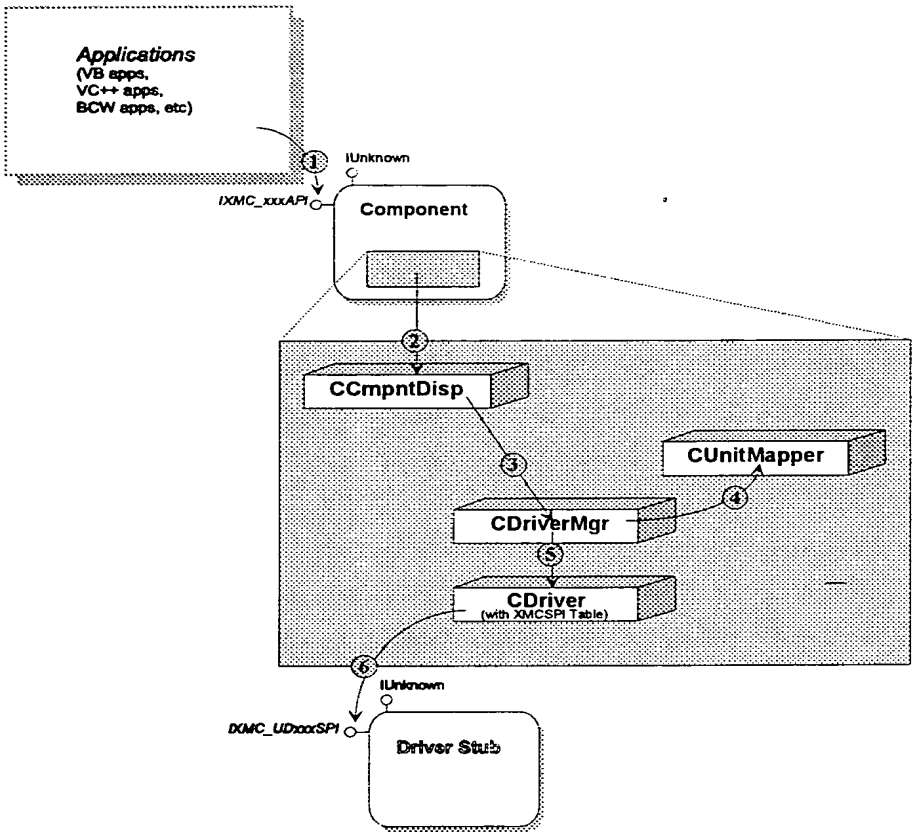
~~Figure 3~~ *Scenario-Map - Initialization.*

FIG. 4



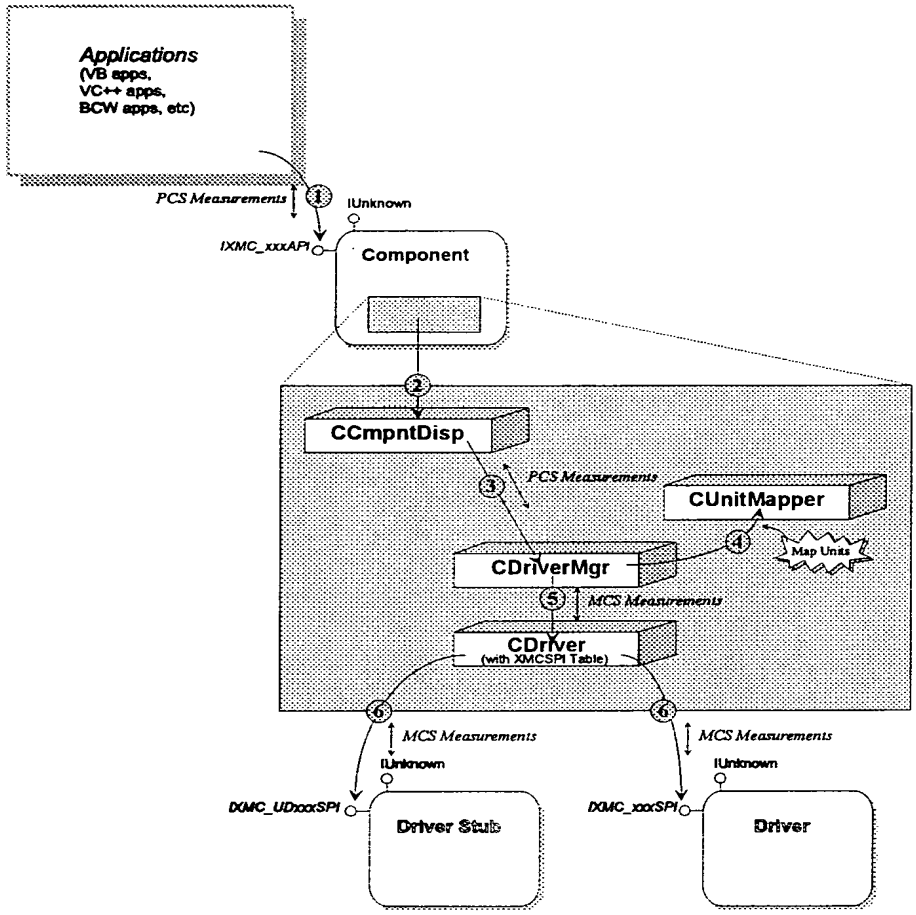
~~Figure 4~~ *Scenario-Map - Core SPI Operation.*

FIG. 5



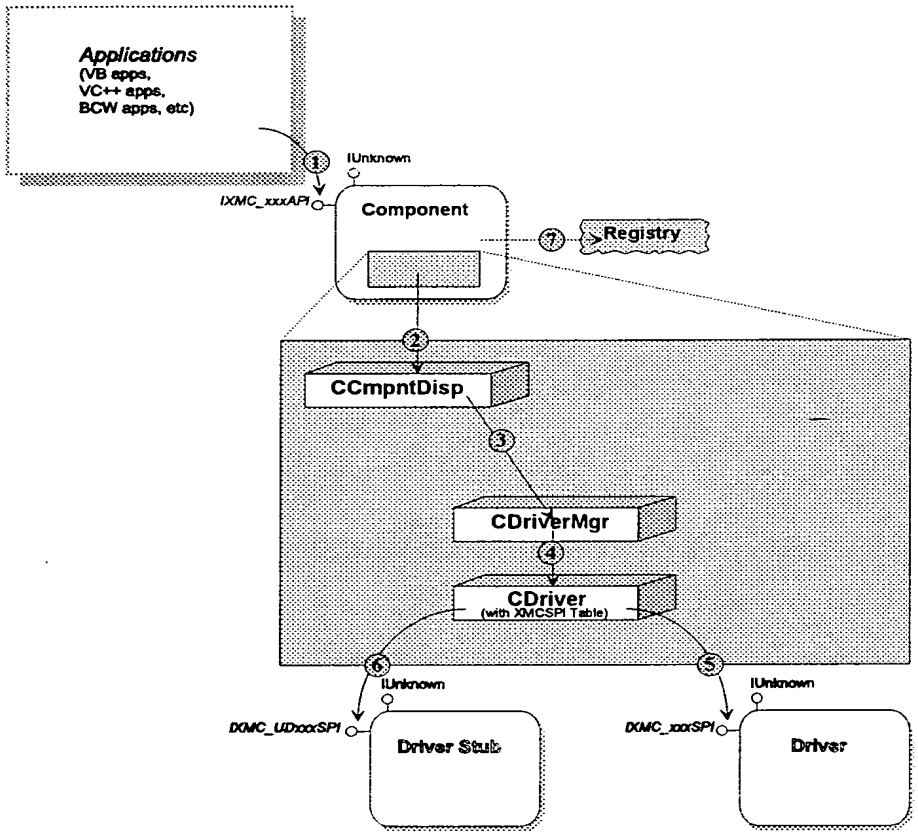
~~Figure 5~~ *Scenario-Map - Extended SPI Operation.*

FIG. 6



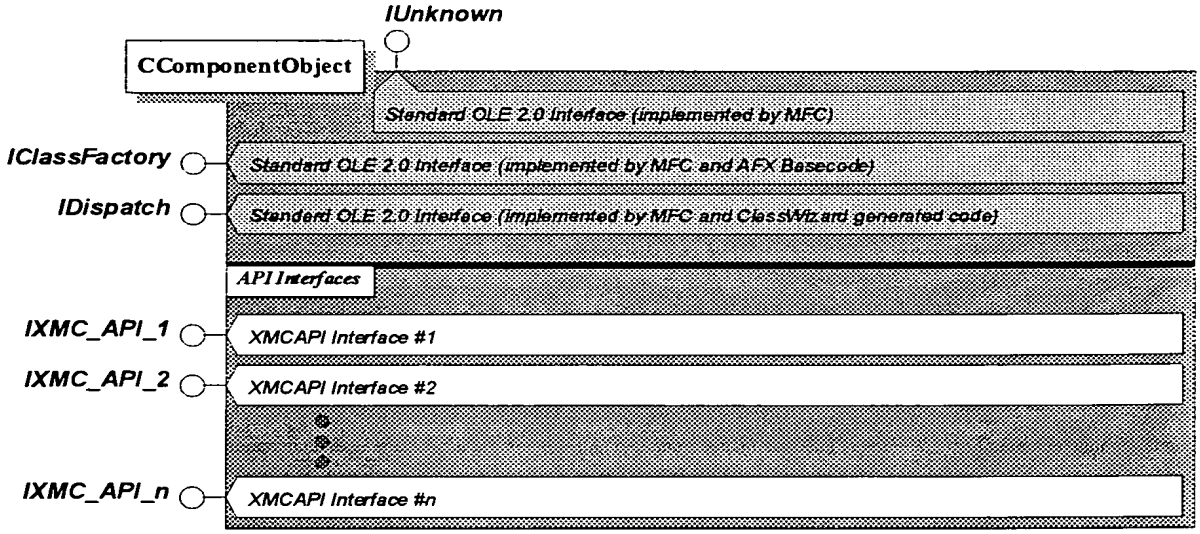
~~Figure 6~~ *Scenario-Map - Unit Mapping.*

FIG. 7



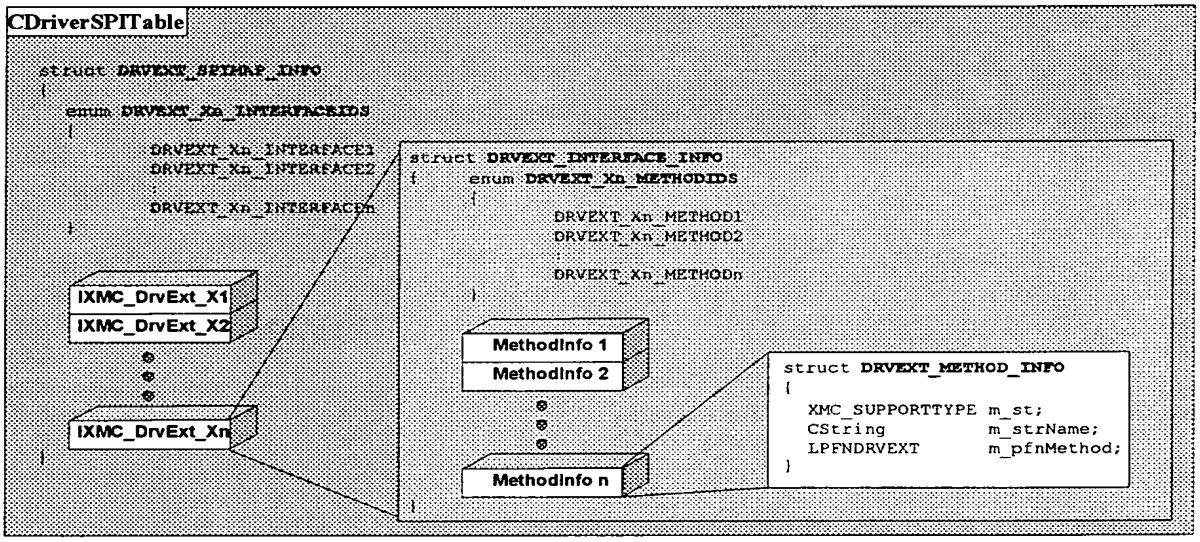
~~Figure 7~~ *Scenario-Map - Clean-up.*

FIG. 8



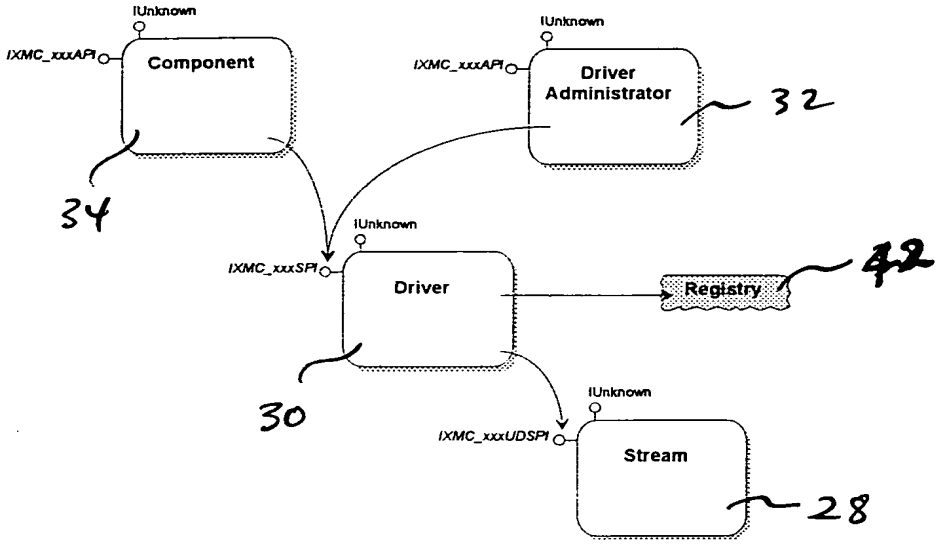
~~Figure 8~~ *Interface-Map.*

FIG. 9



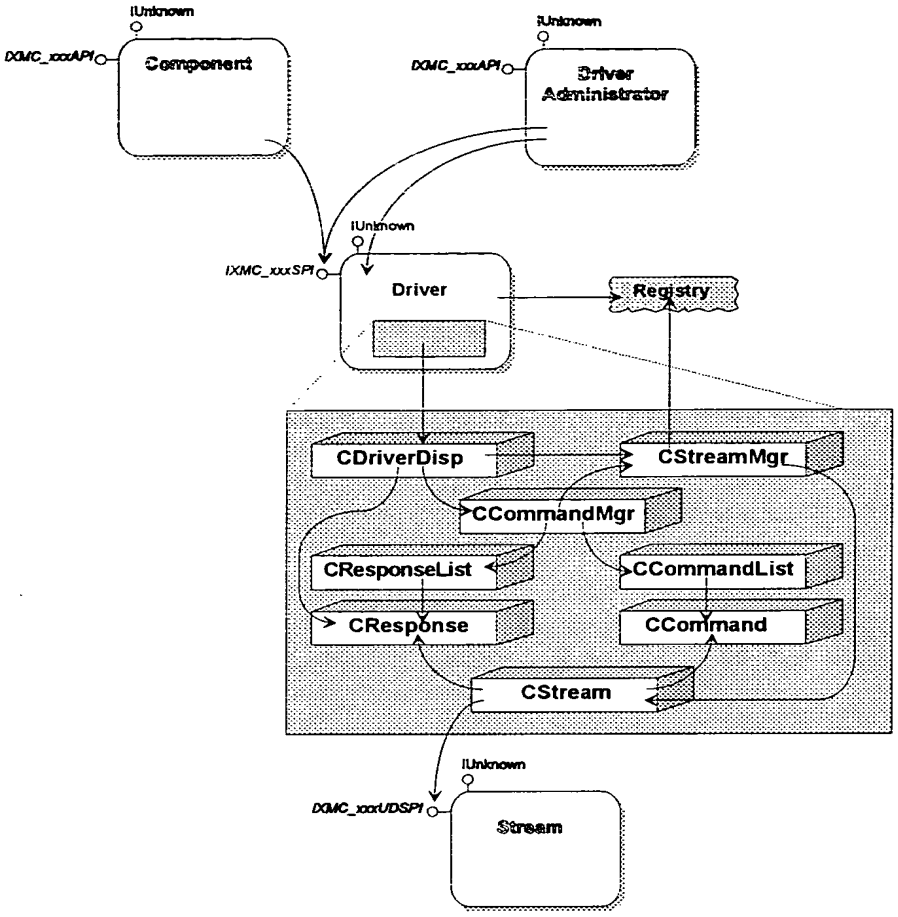
~~Figure 9~~ *Data-Map - **CDriver** class with XMCSPI table.*

FIG. 10



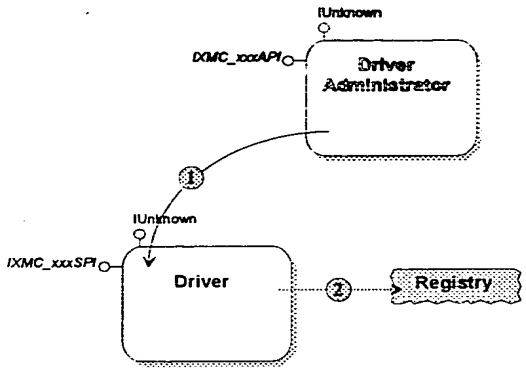
~~Figure 1~~ *Module Interaction-Map.*

FIG. 11



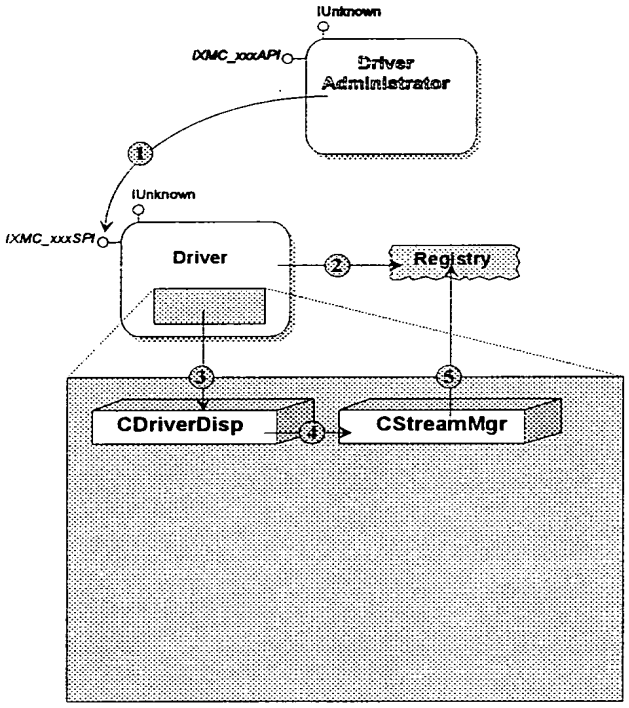
~~Figure 2~~ *Object Interaction-Map.*

FIG. 12



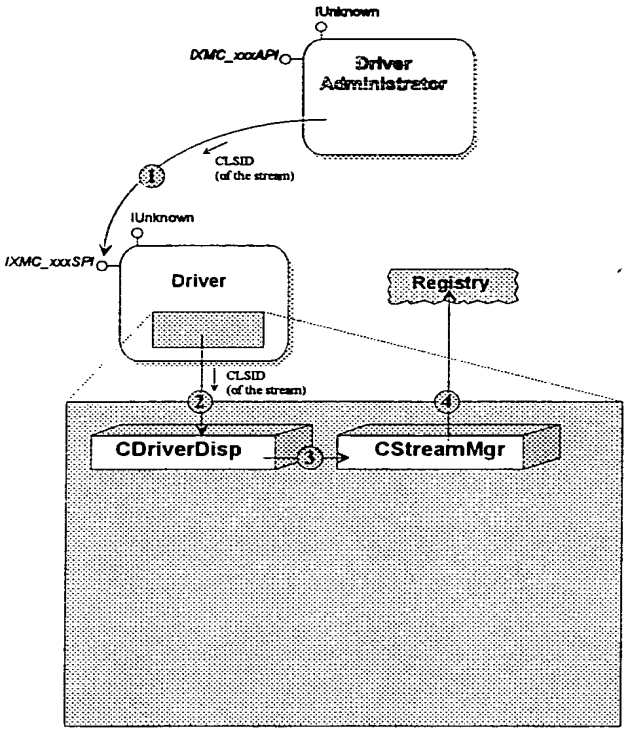
~~Figure 3~~ *Scenario-Map - Registration.*

FIG. 13



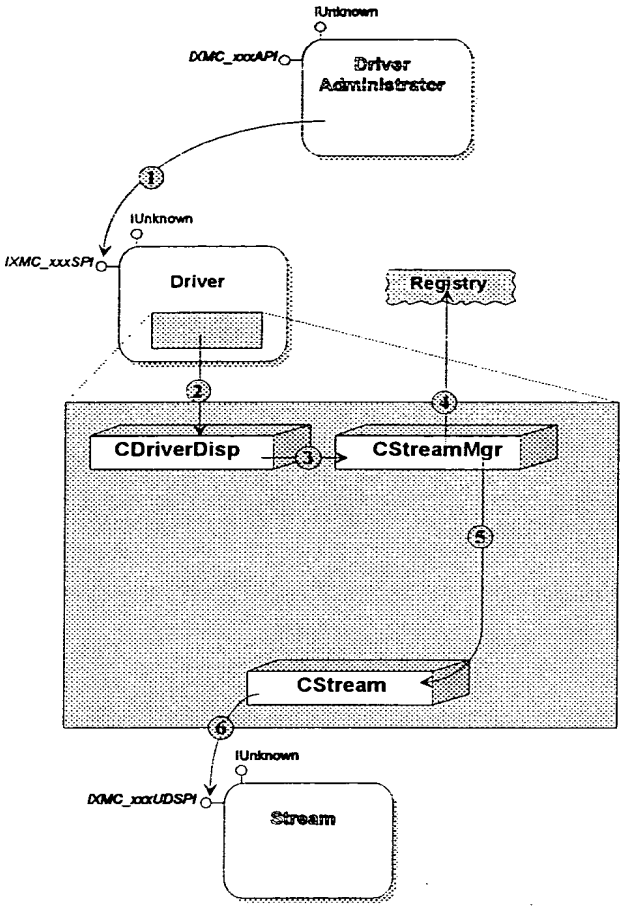
~~Figure 4~~ *Scenario-Map - Initialization by Driver Administrator.*

FIG. 14



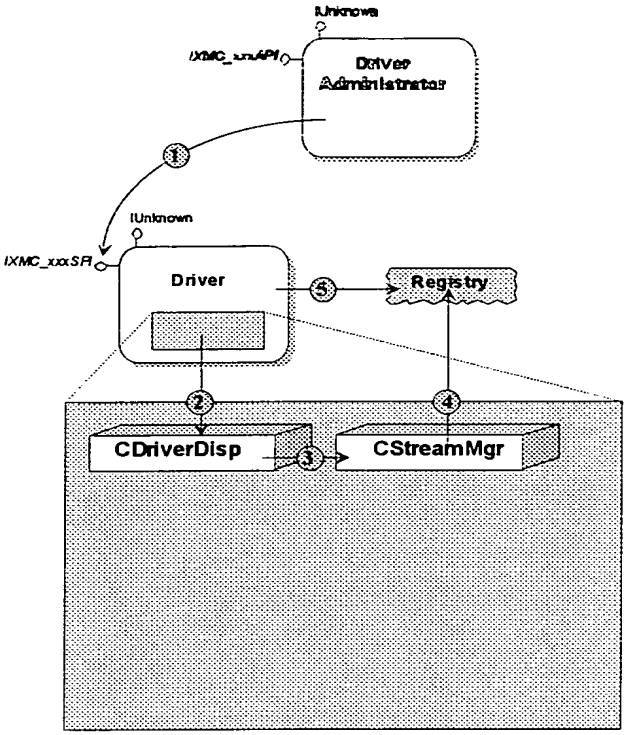
~~Figure 5~~ *Scenario-Map - Adding a Stream.*

FIG. 15



~~Figure 6~~ *Scenario-Map - Query Operation.*

FIG. 16



~~Figure 7~~ *Scenario-Map - Clean-up by Driver Administrator.*

FIG. 17

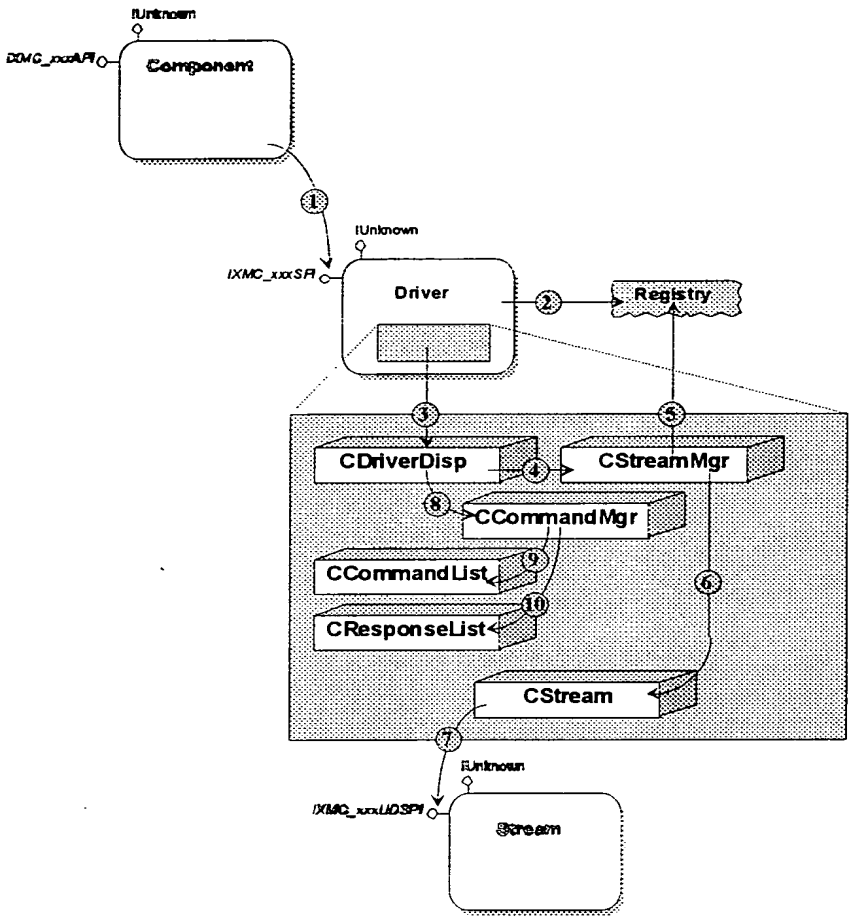
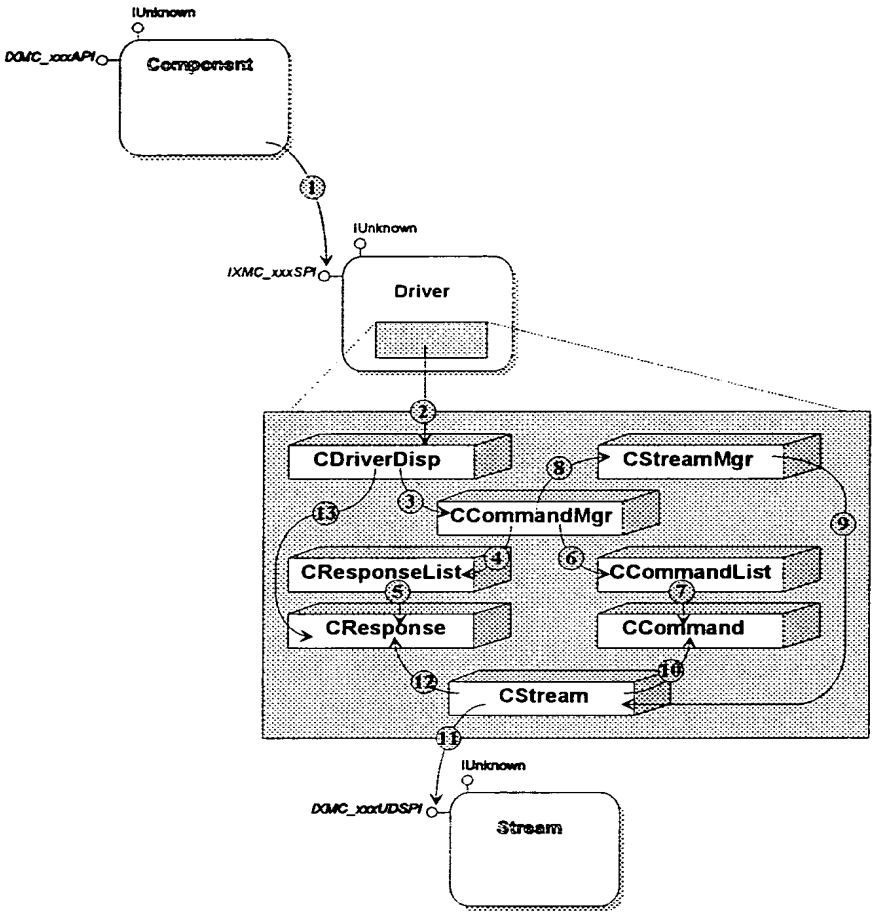


Figure 8 *Scenario-Map - Initialization by Component.*

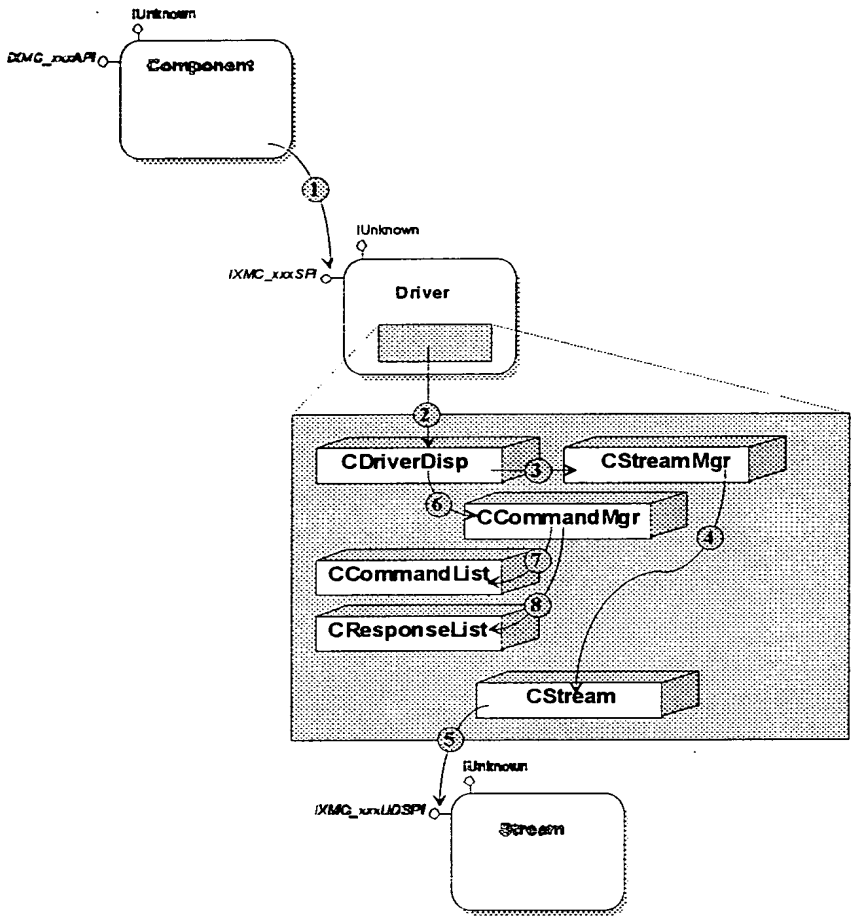
FIG. 18

08/454736



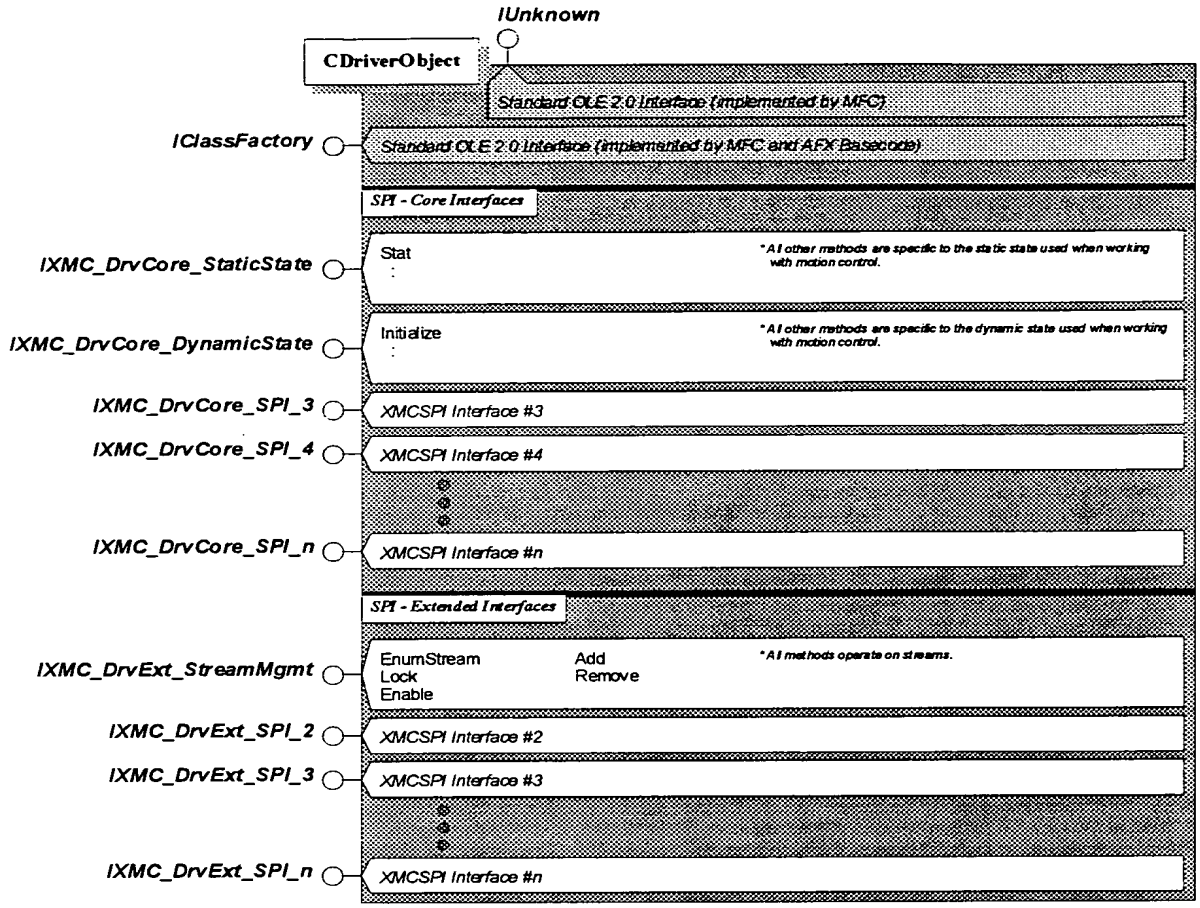
~~Figure 9~~ *Scenario-Map - Command Operations.*

FIG. 19



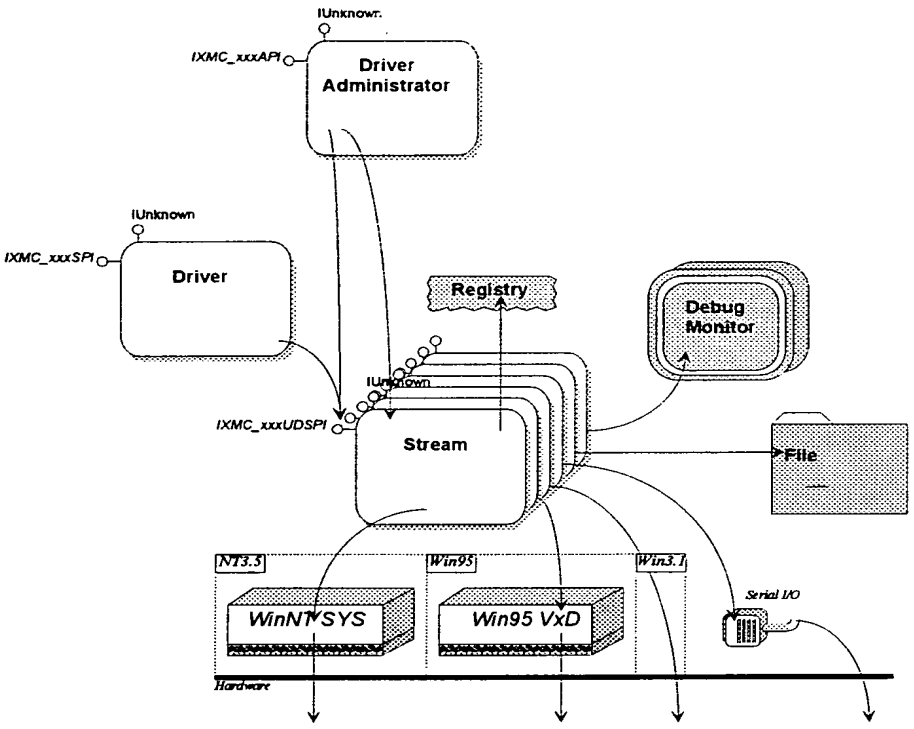
~~Figure 18~~ *Scenario-Map - Clean-up by Component.*

FIG. 20



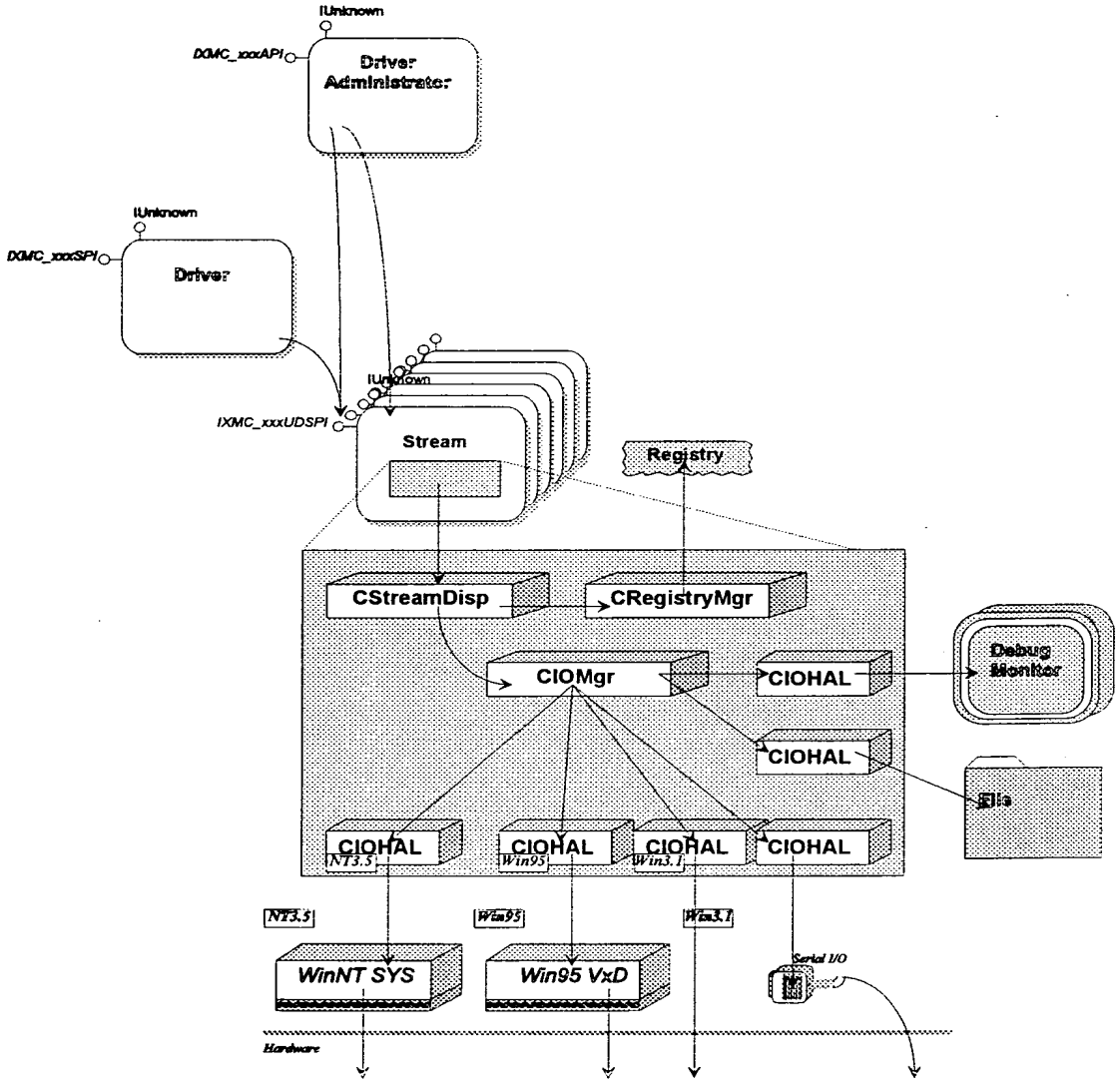
~~Figure 11~~ *Interface-Map.*

FIG. 21



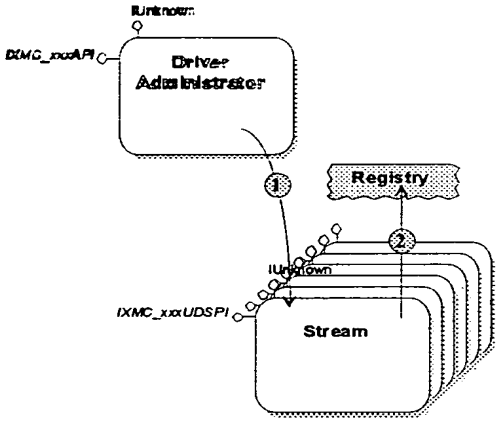
~~Figure 1~~ *Module IA-Map.*

FIG. 22



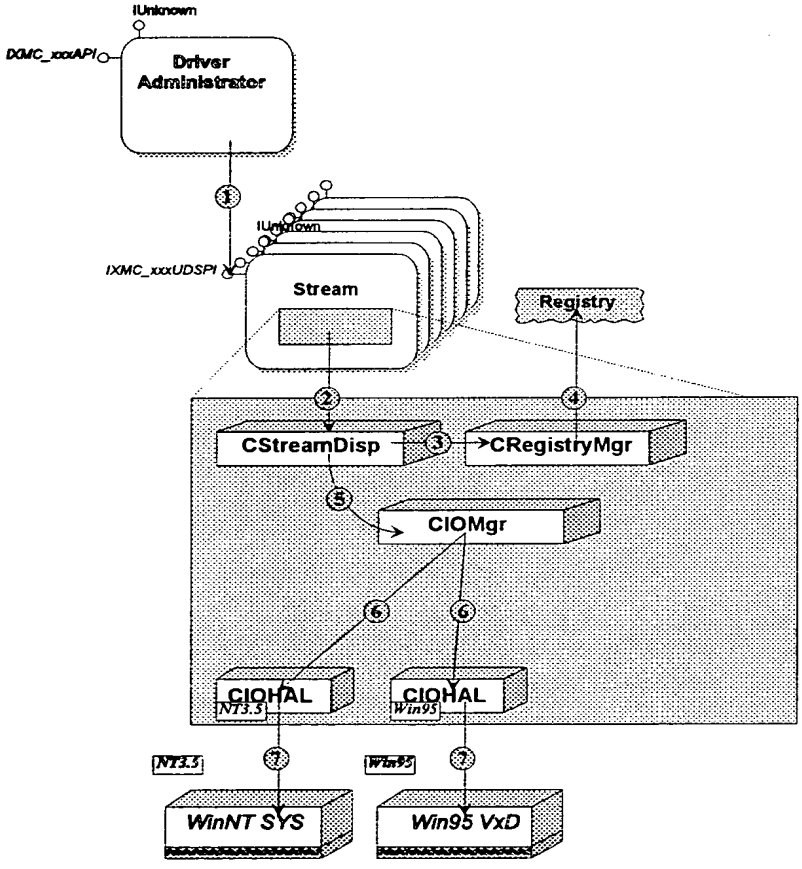
~~Figure 2~~ *Object IA-Map.*

FIG. 23



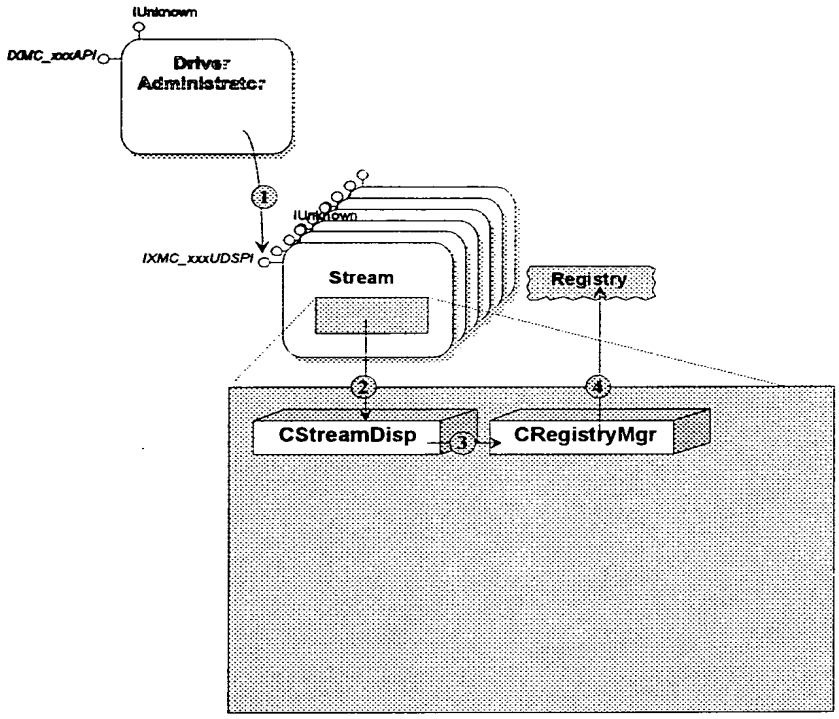
~~Figure 3~~ *Scenario-Map - Registration*

FIG. 24



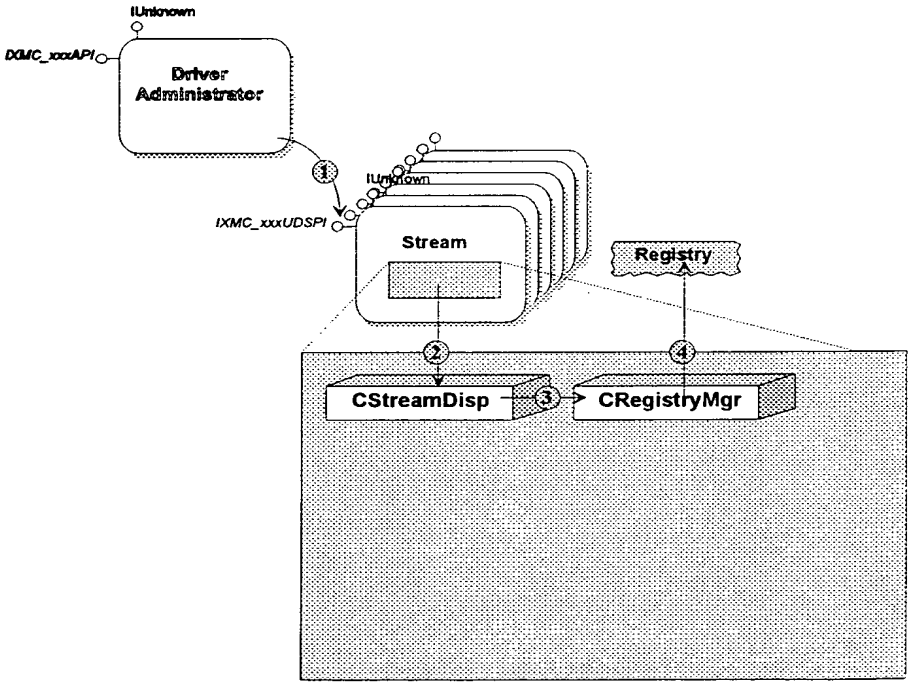
~~Figure 4~~ *Scenario-Map - Initialization*

FIG. 25



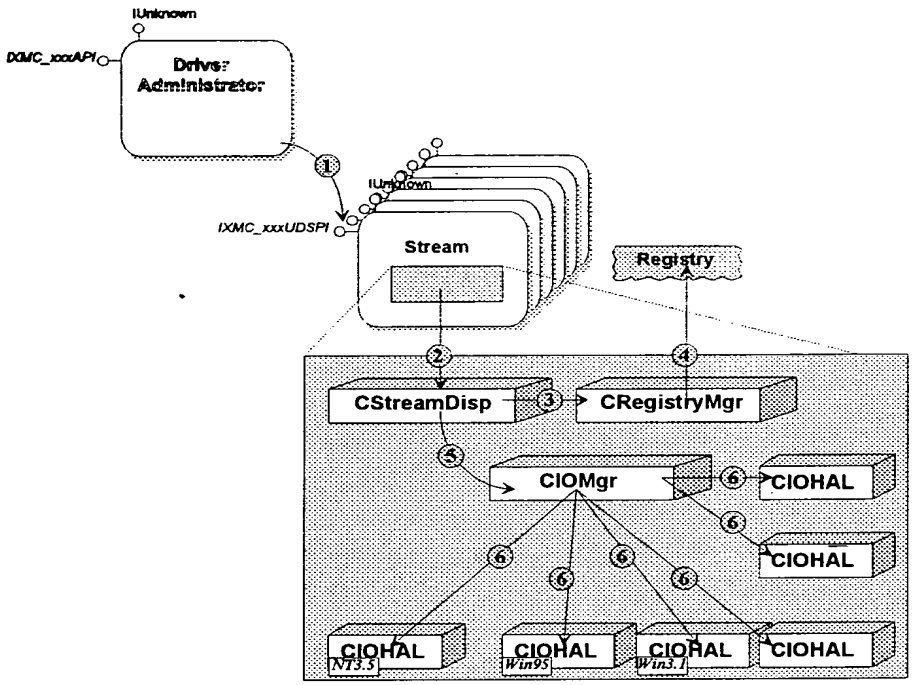
~~Figure 5~~ *Scenario-Map - Setup*

FIG. 26



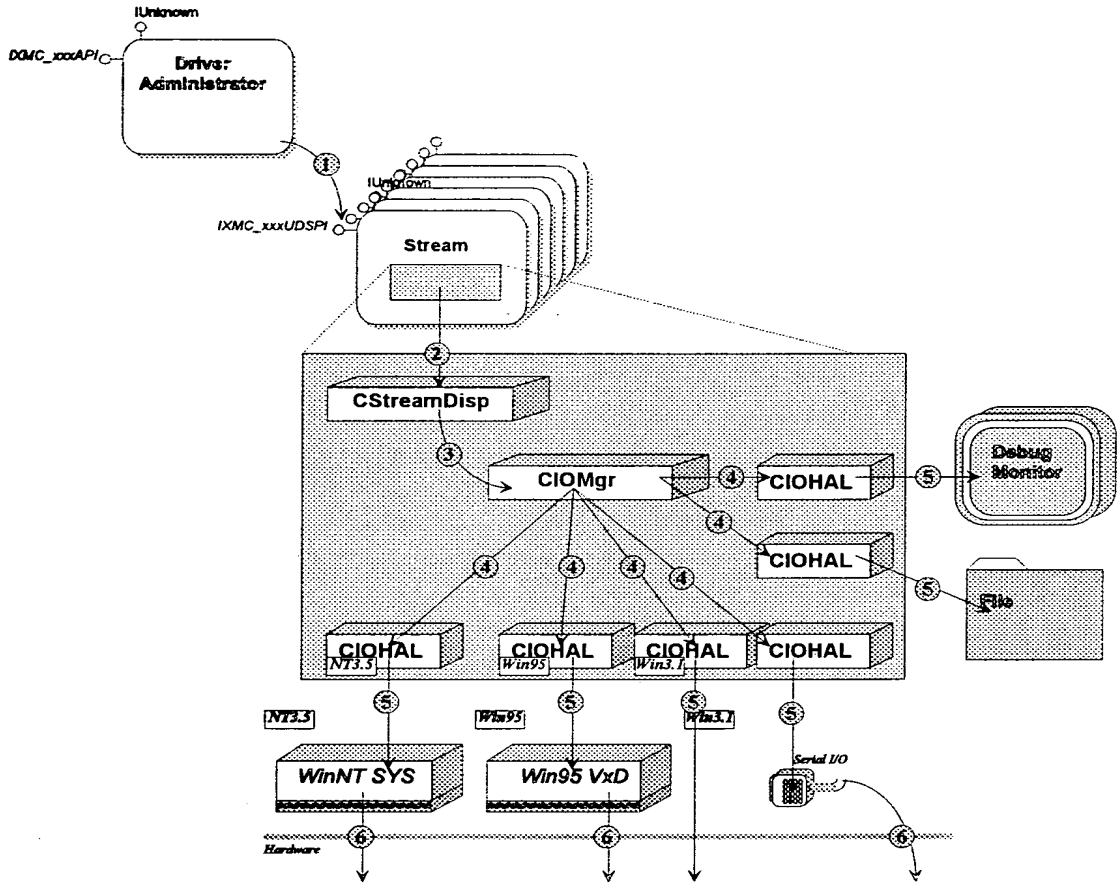
~~Figure 6~~ *Scenario-Map - Clean-up*

FIG. 27



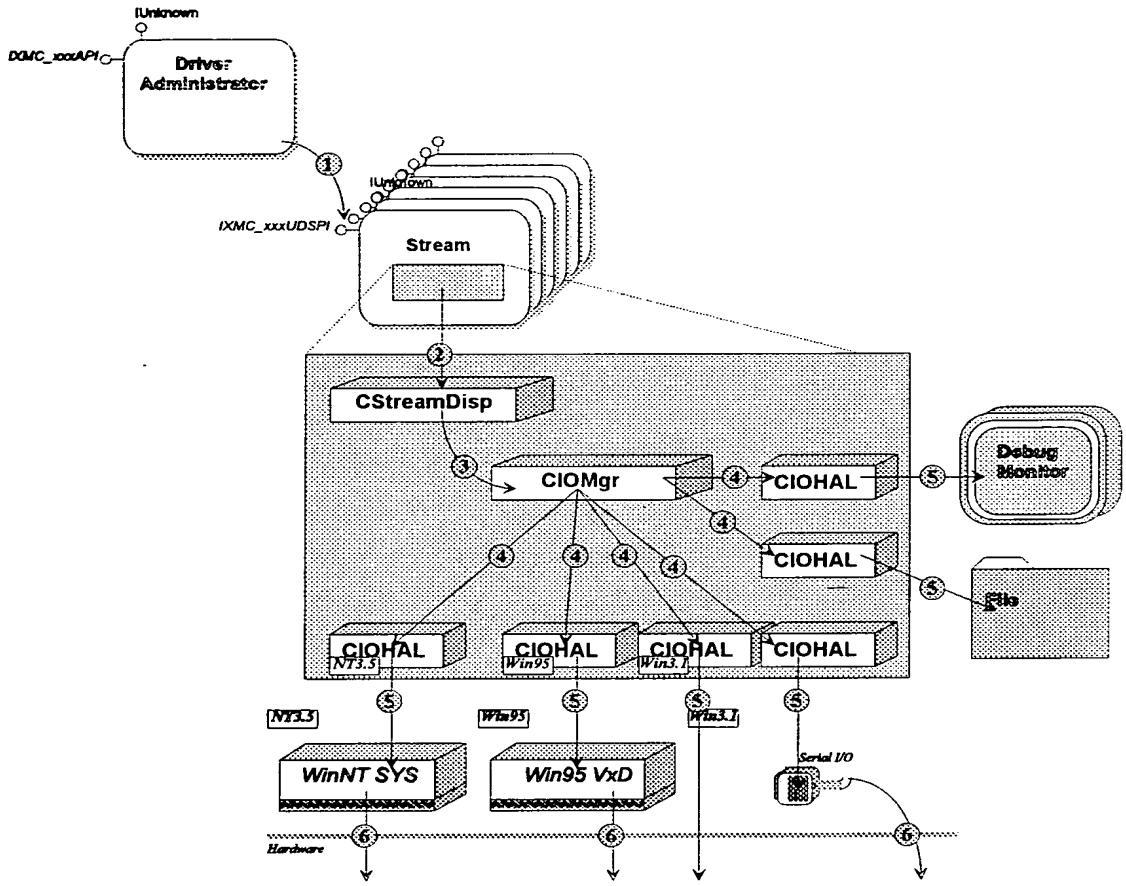
~~Figure 7~~ *Scenario-Map - Initialization*

FIG. 28



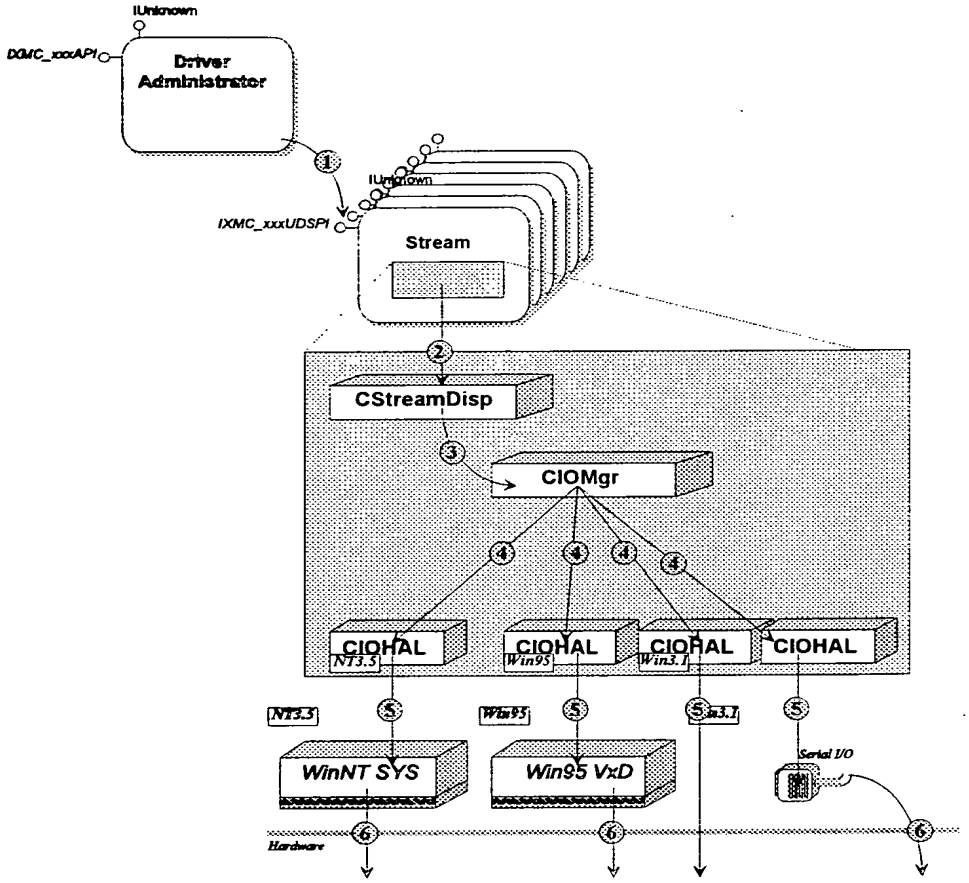
~~Figure 8~~ *Scenario-Map - Opening the Stream*

FIG. 29



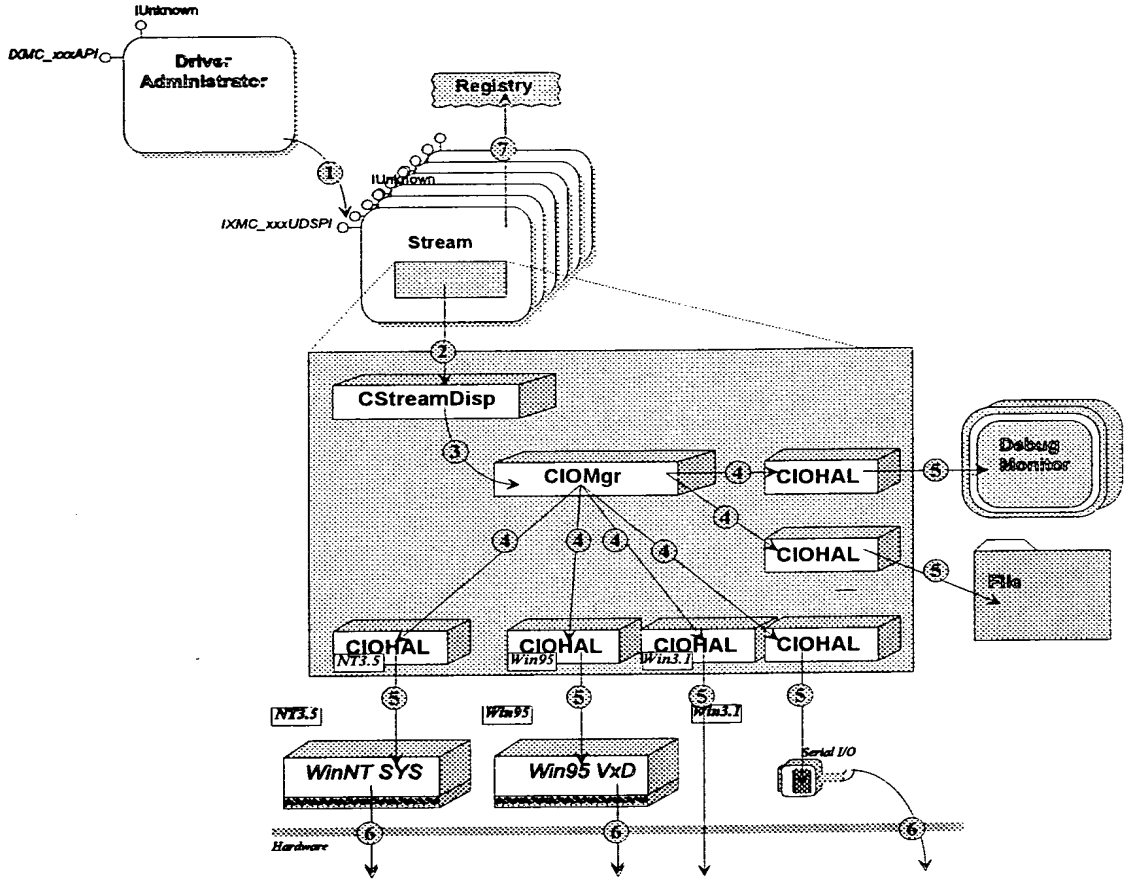
~~Figure 9~~ *Scenario-Map - Writing Data.*

FIG. 30



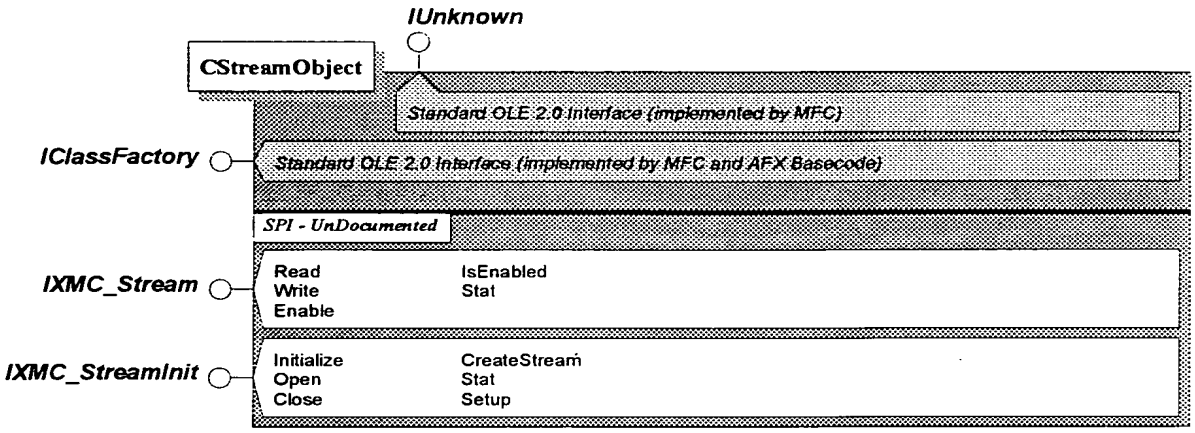
~~Figure 10~~ *Scenario-Map - Reading Data.*

FIG. 31



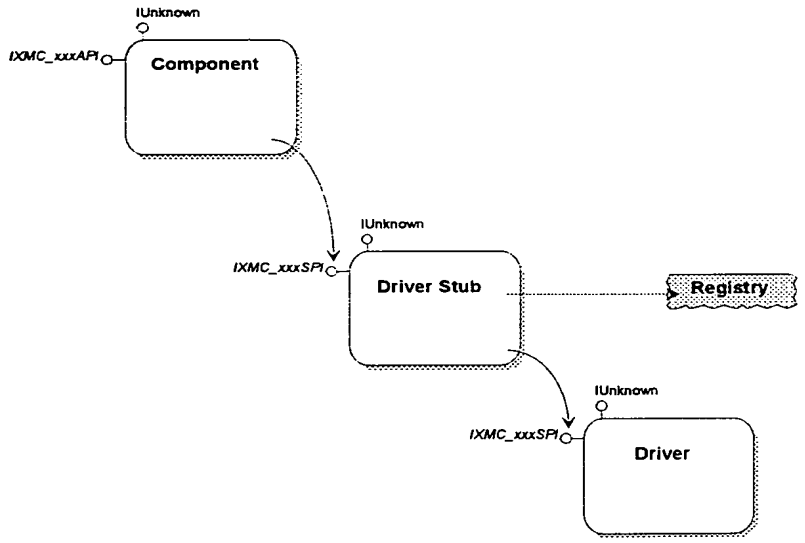
~~Figure 11~~ *Scenario-Map - Clean-up.*

FIG. 32



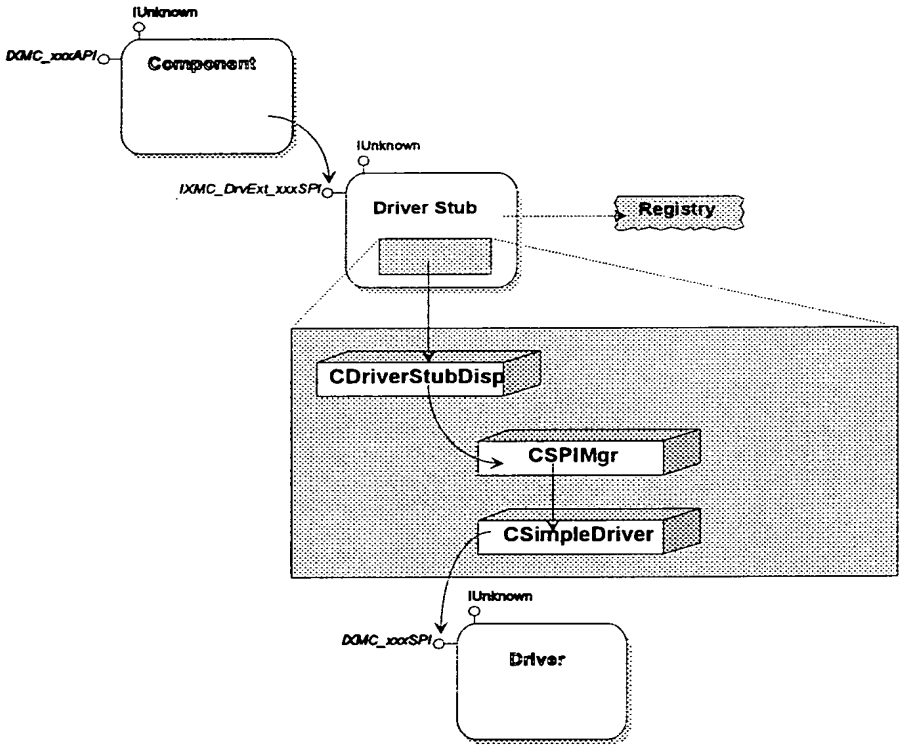
~~Figure 12~~ *Interface-Map.*

FIG. 33



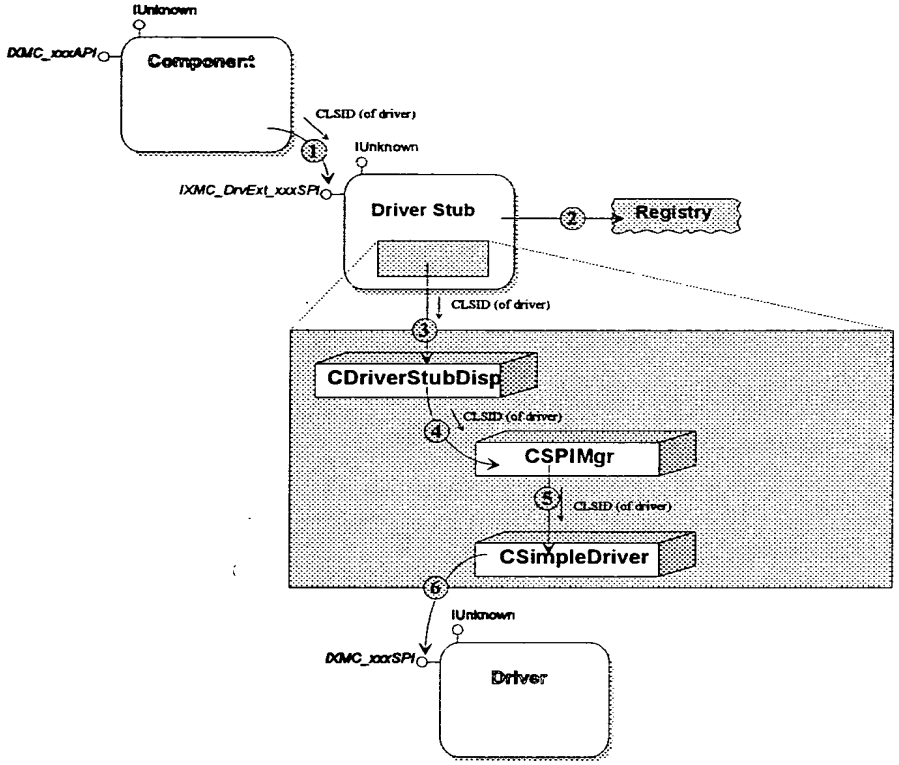
~~Figure 1~~ *Module Interaction-Map.*

FIG. 34



~~Figure 2~~ *Object Interaction-Map.*

FIG. 35



~~Figure 3~~ *Scenario-Map - Initialization.*

FIG. 36

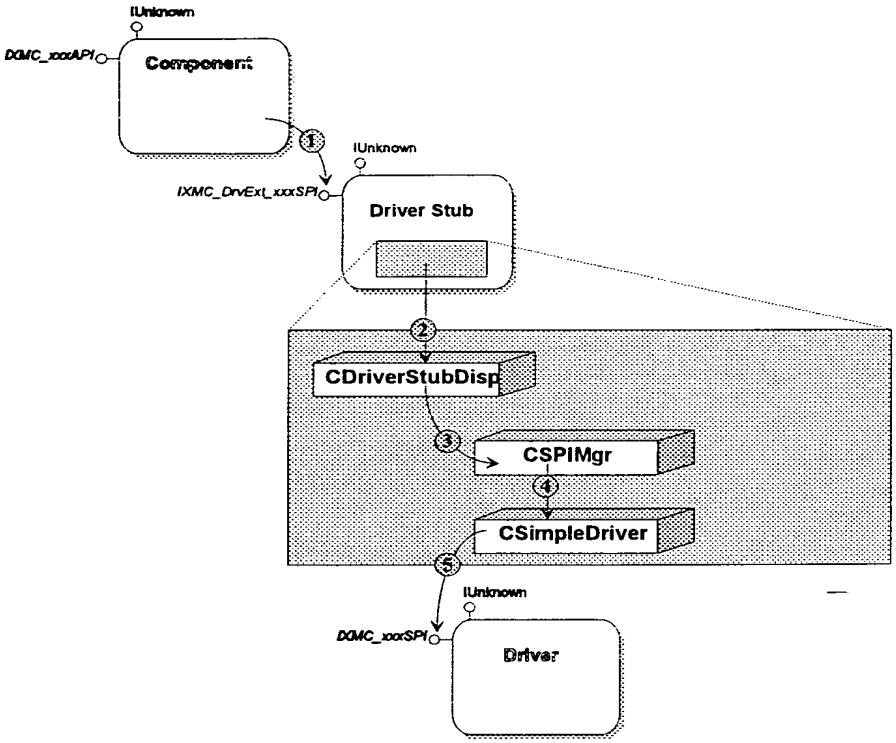
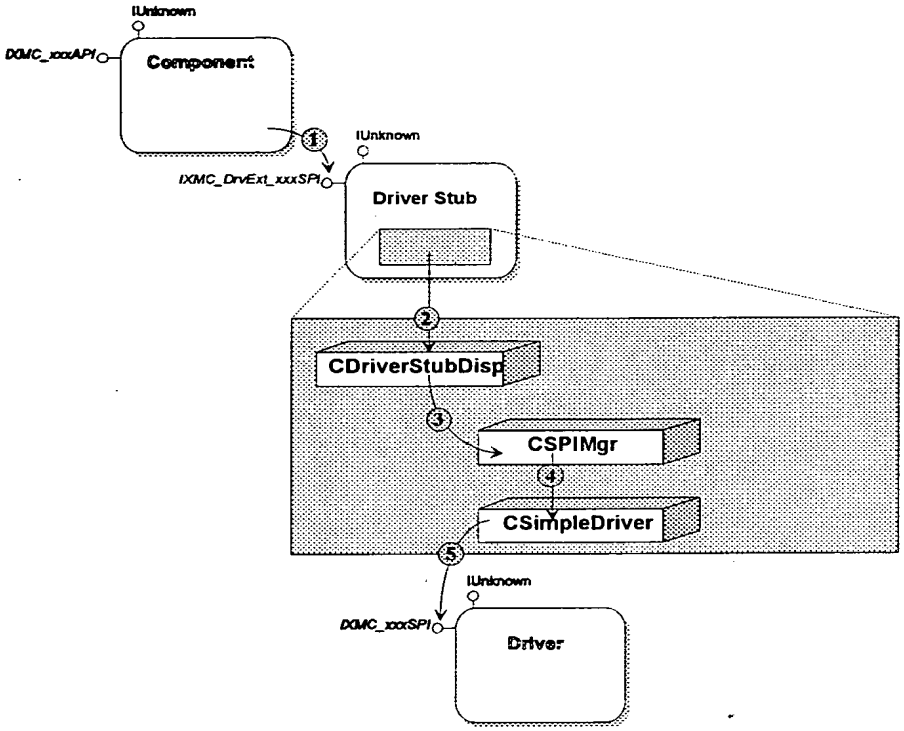


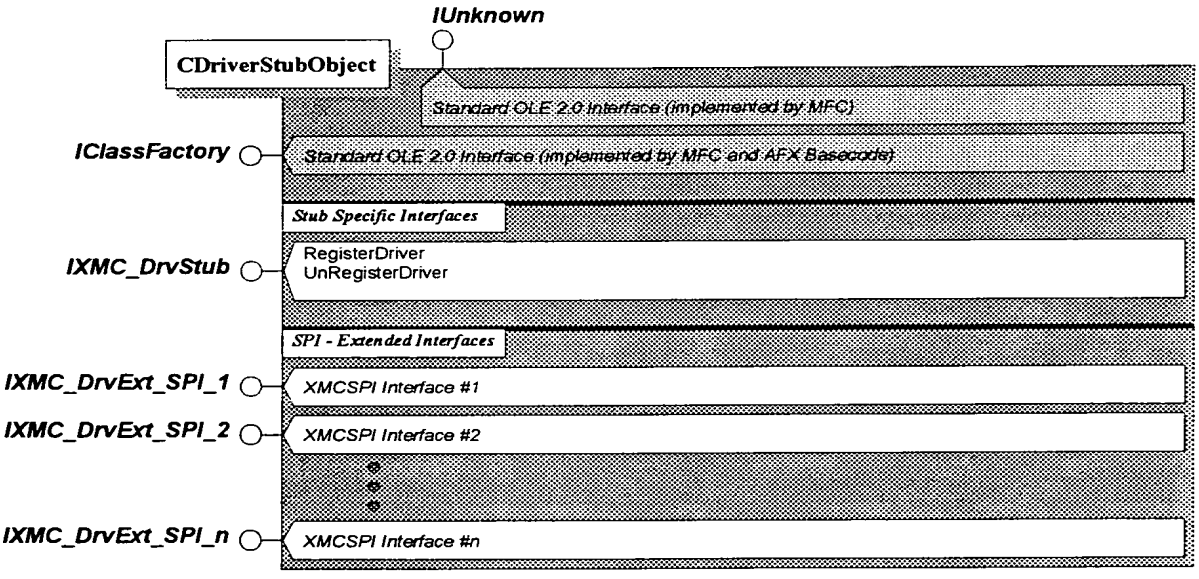
Figure 4 *Scenario-Map - Operations.*

FIG. 37



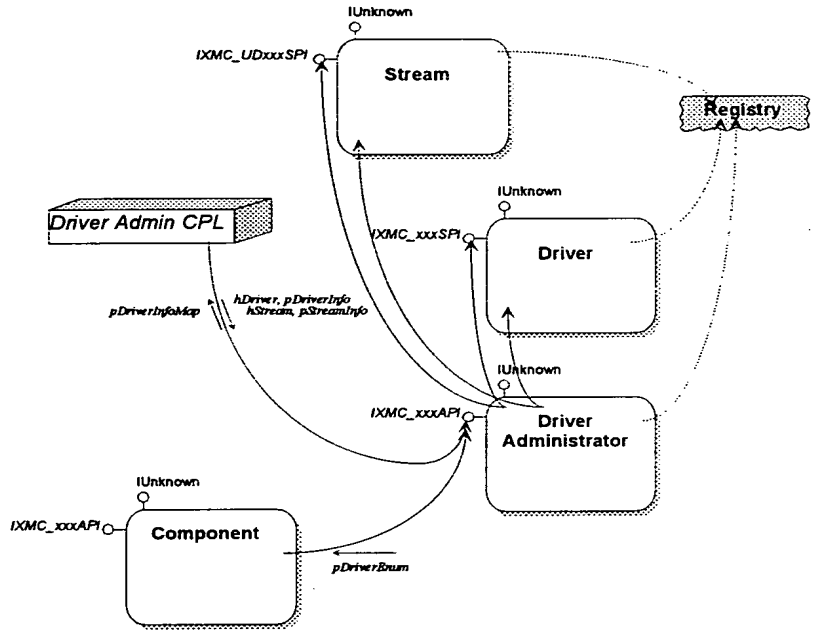
~~Figure 5~~ *Scenario-Map - Clean-up.*

FIG. 38



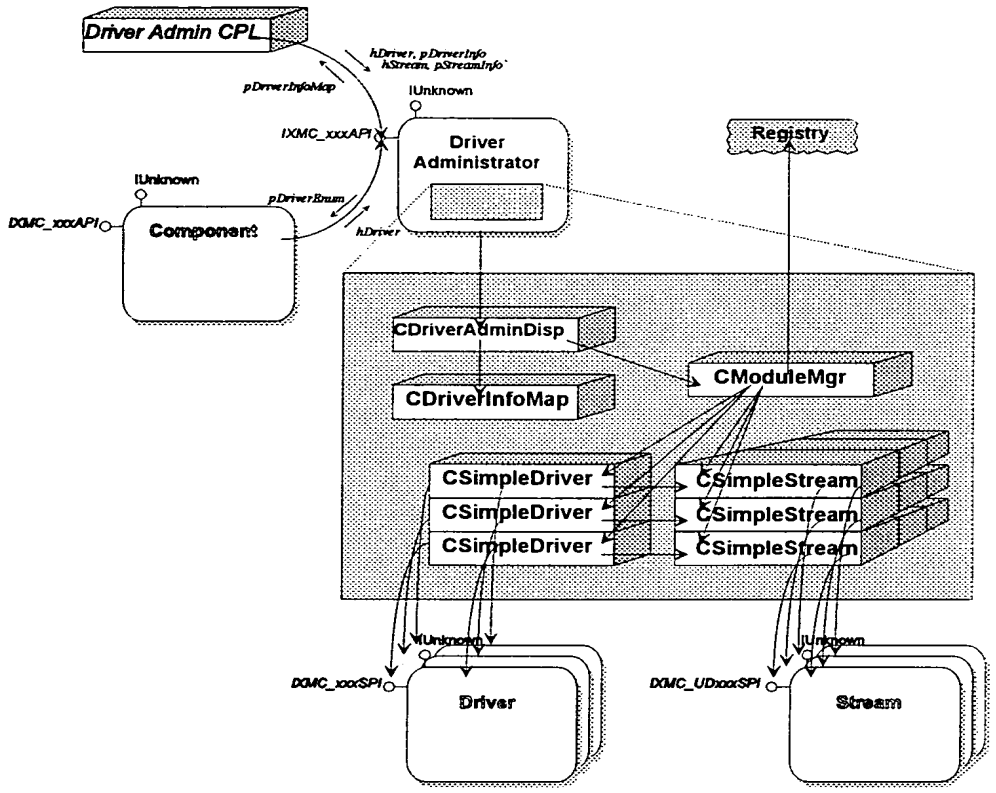
~~Figure 6~~ *Interface-Map.*

FIG. 39



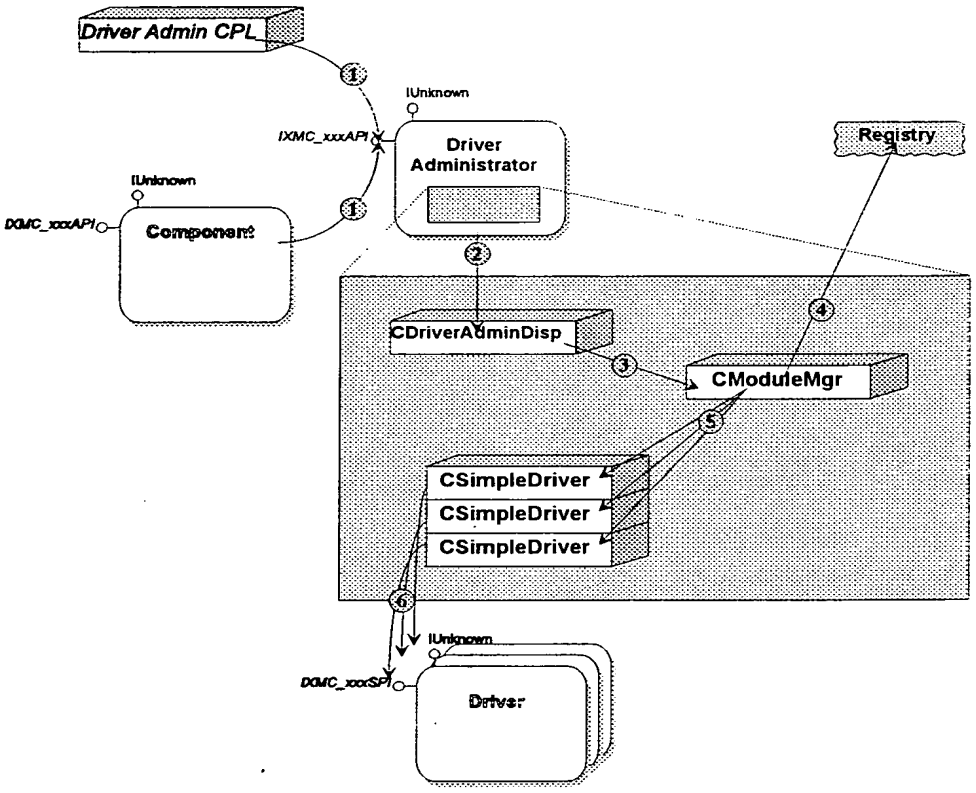
~~Figure 1~~ *Module Interaction-Map.*

FIG. 40



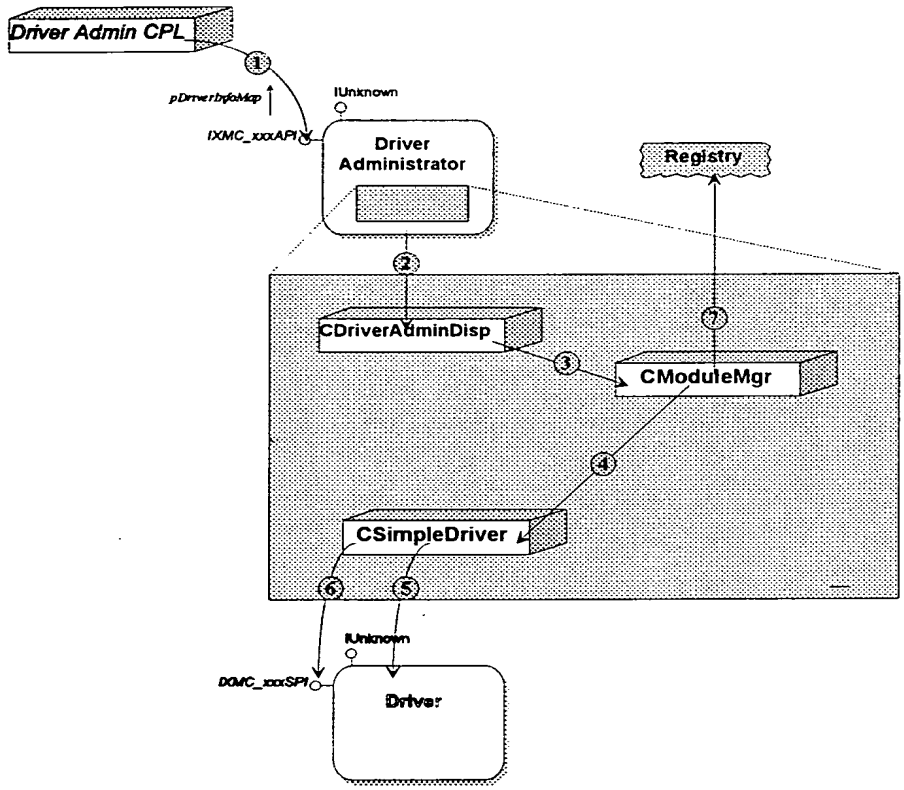
~~Figure 2~~ *Object Interaction-Map.*

FIG. 41



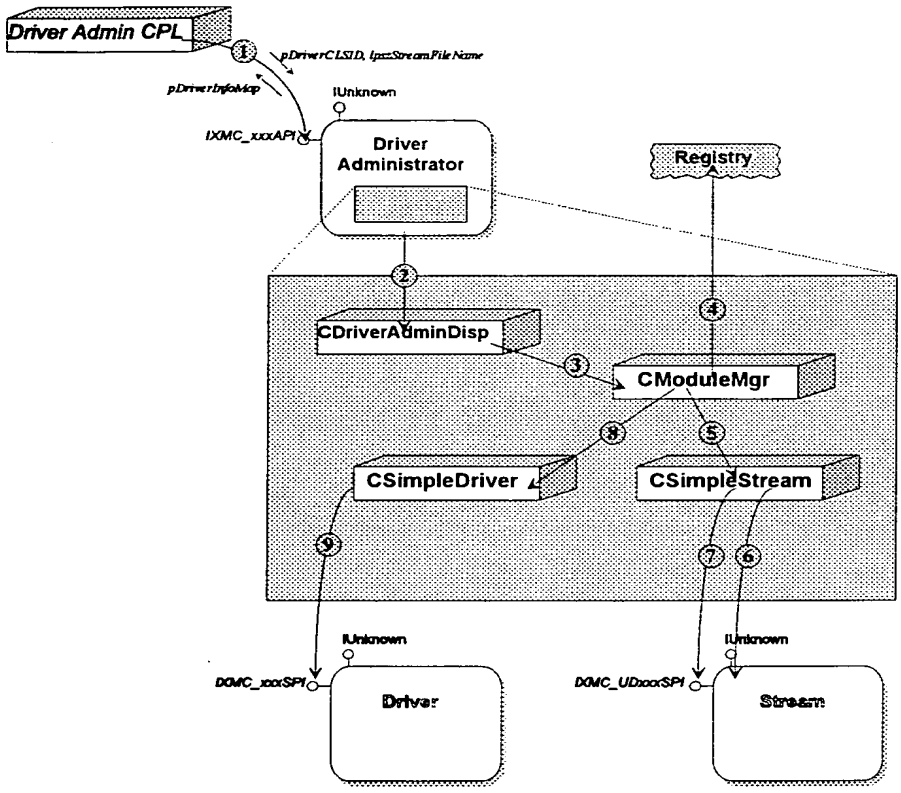
~~Figure 3~~ *Scenario-Map - Initialization.*

FIG. 42



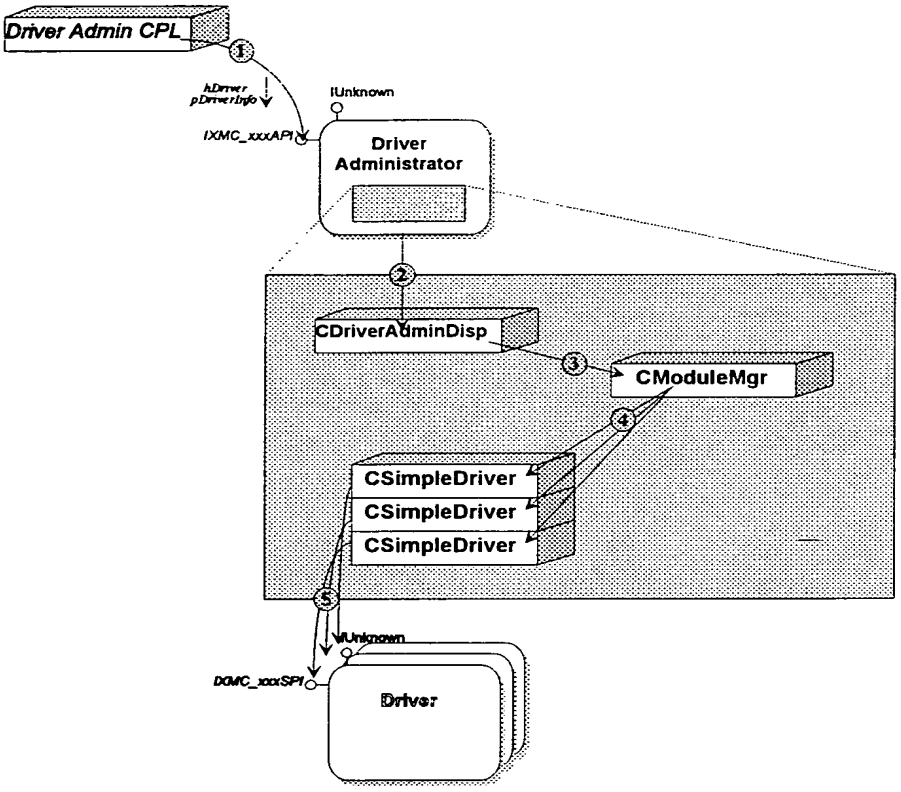
~~Figure 4~~ *Scenario-Map - Registering a Driver.*

FIG. [redacted]
43



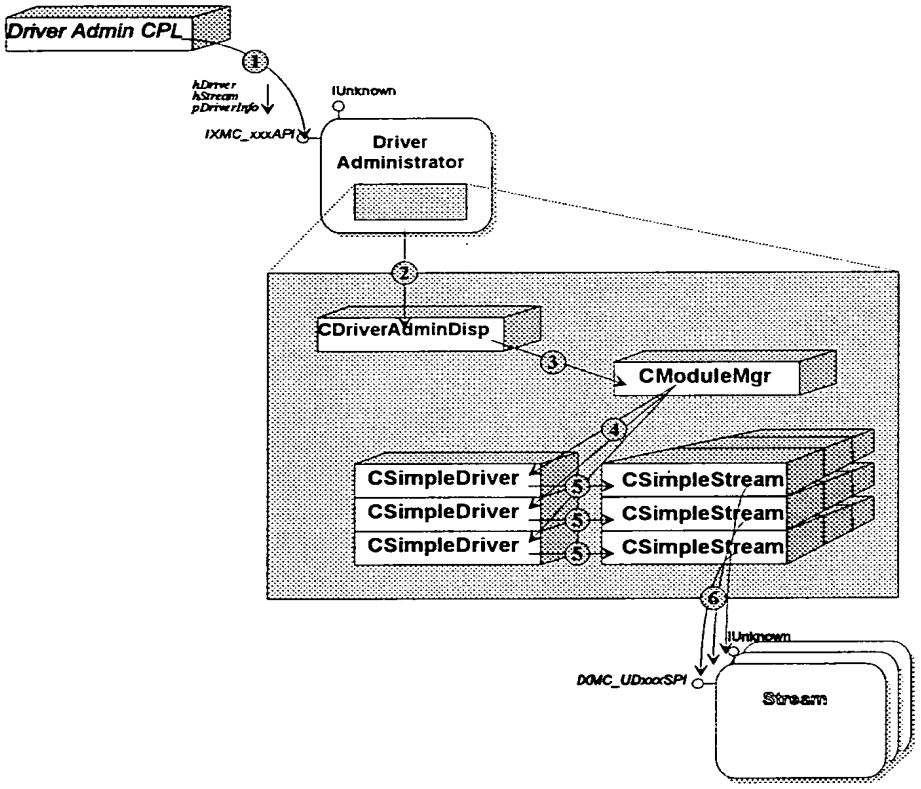
~~Figure 5~~ *Scenario-Map - Registering a Stream.*

FIG. 44



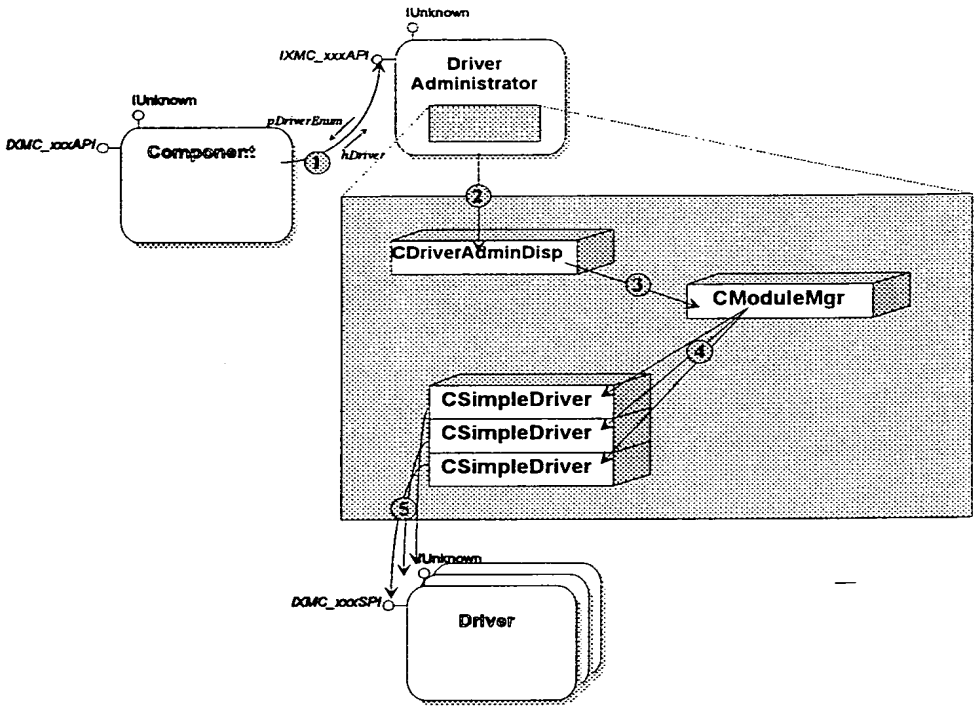
~~Figure 6~~ *Scenario-Map - Setting Driver Information.*

FIG. 45



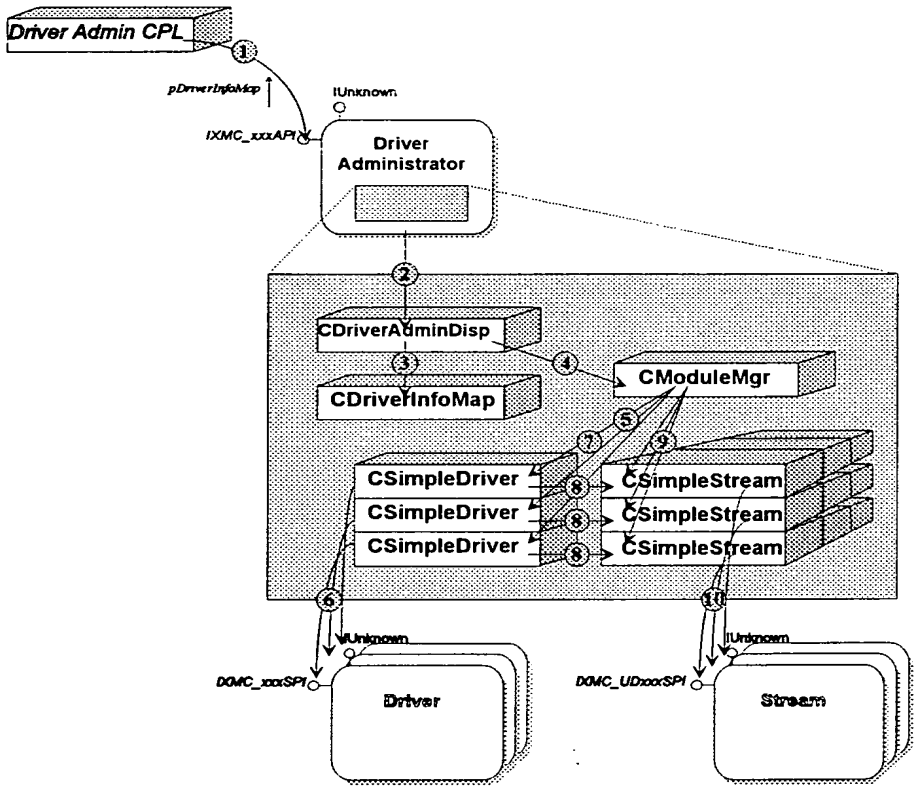
~~Figure 7~~ *Scenario-Map - Setting Stream Information.*

FIG. 46



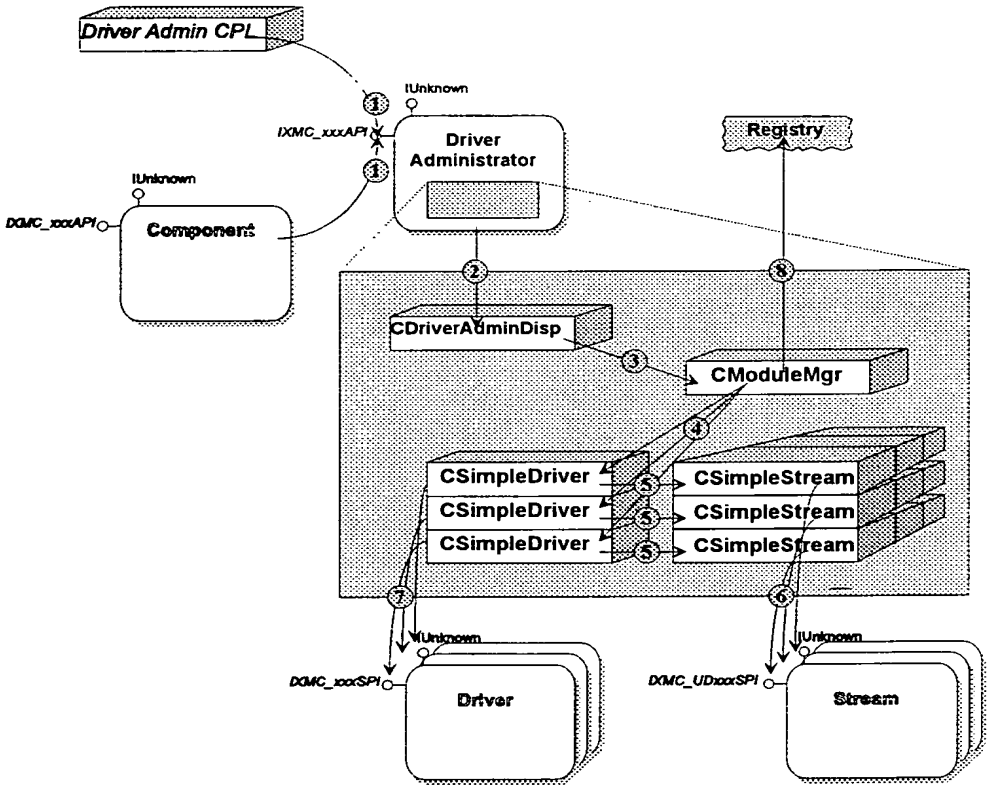
~~Figure 8~~ *Scenario-Map - Querying the Driver Enumeration.*

FIG. ~~4A~~ 47



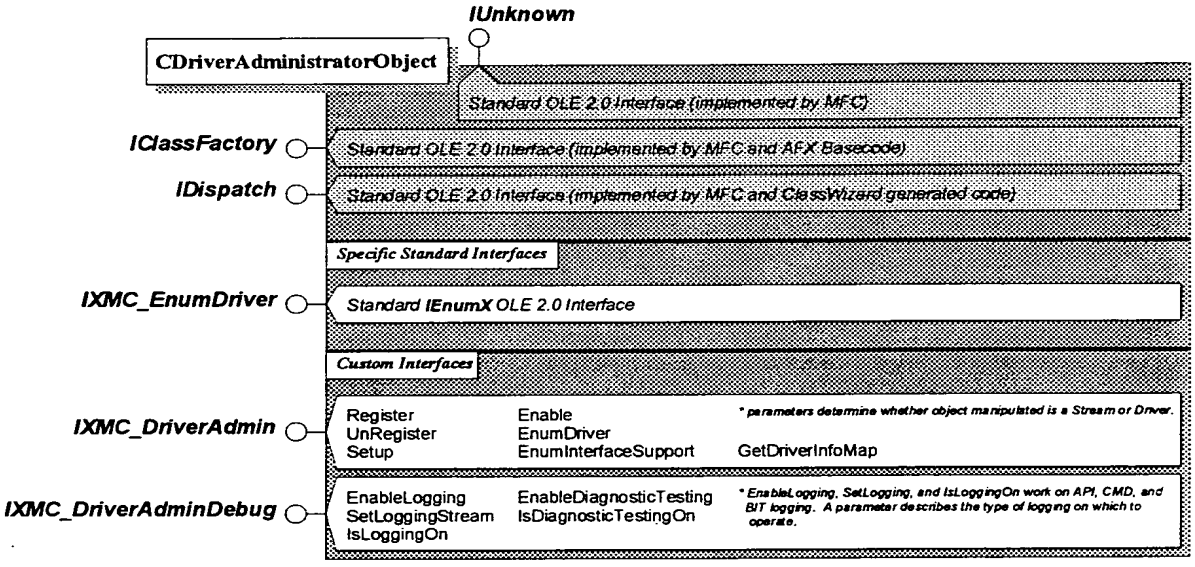
~~Figure 9~~ *Scenario-Map - Querying the Driver Info Map.*

FIG. 48



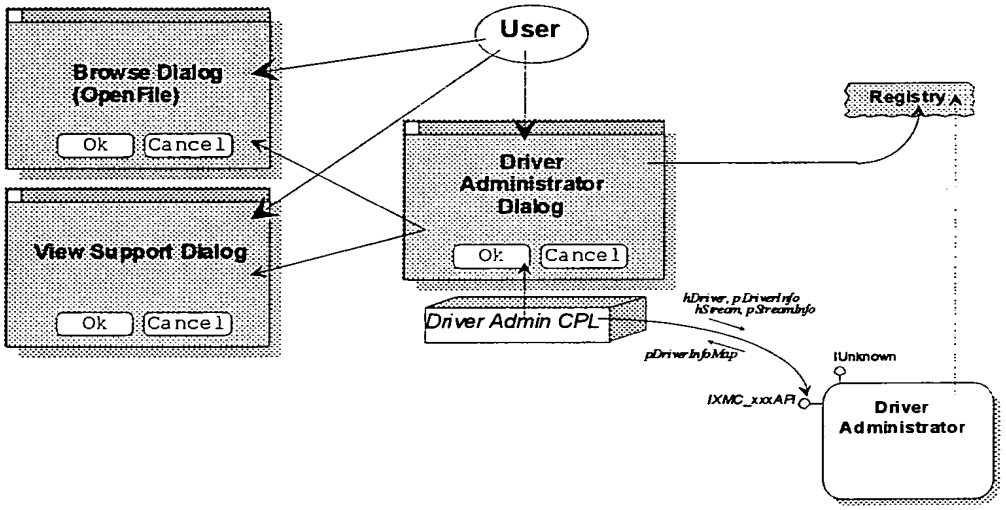
~~Figure 10~~ *Scenario-Map - Clean-up.*

FIG. 49



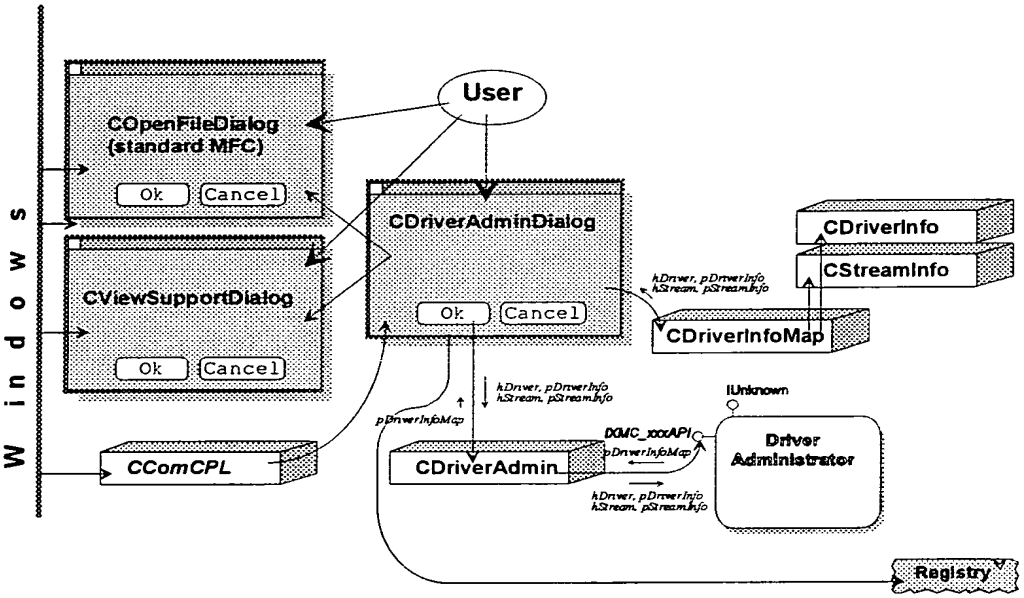
~~Figure 11~~ *Interface-Map.*

FIG. 50



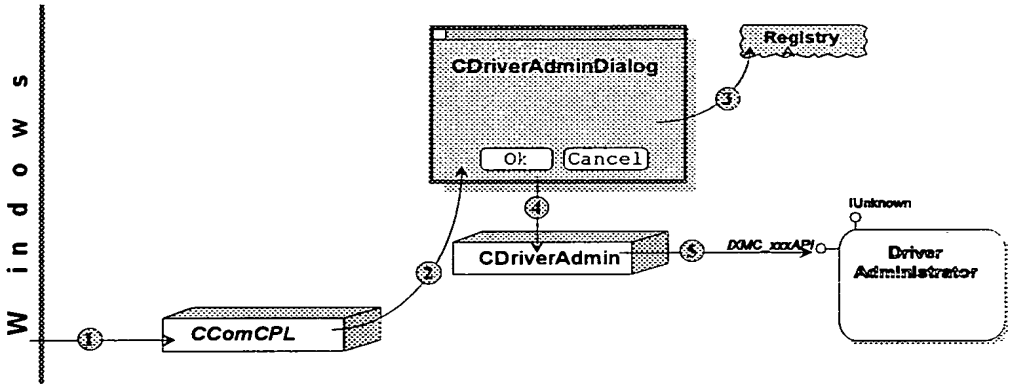
~~Figure 1~~ *Module IA-Map.*

FIG. 51



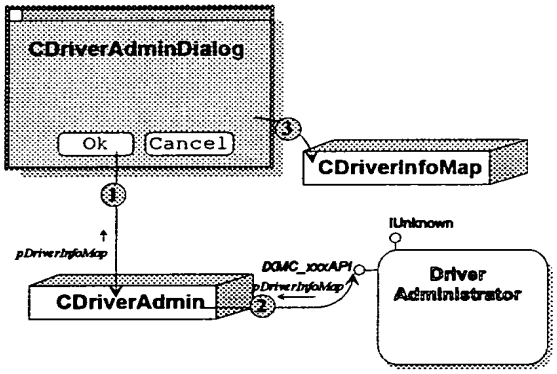
~~Figure 2~~ *Object IA-Map.*

FIG. 52



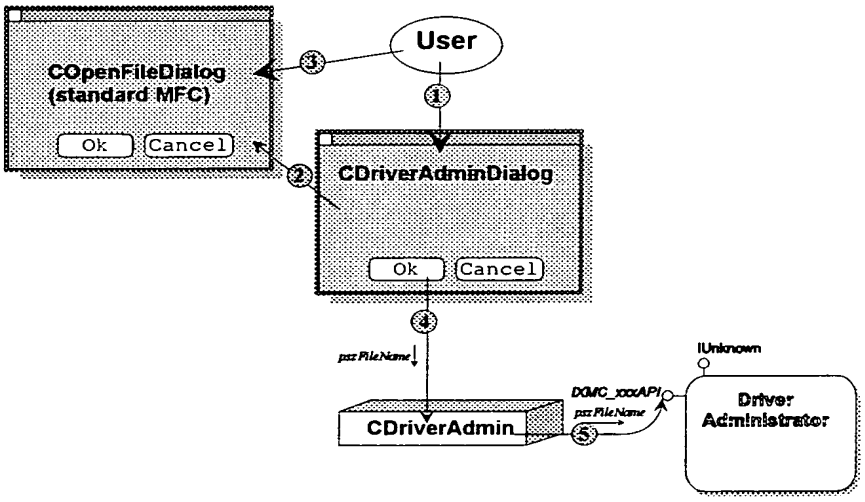
~~Figure 3~~ *Initializing the Application.*

FIG. 53



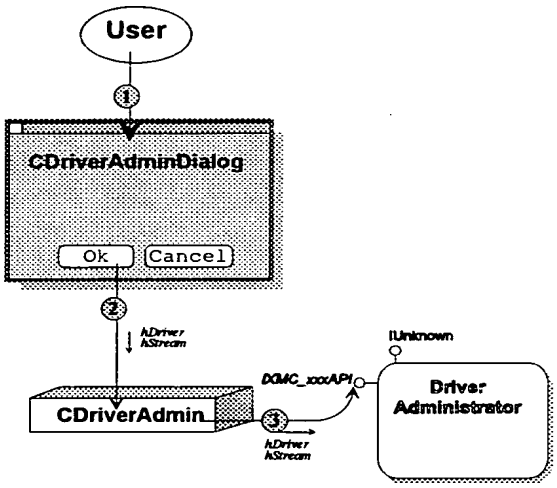
~~Figure 4~~ *Main Dialog Initialization.*

FIG. 54



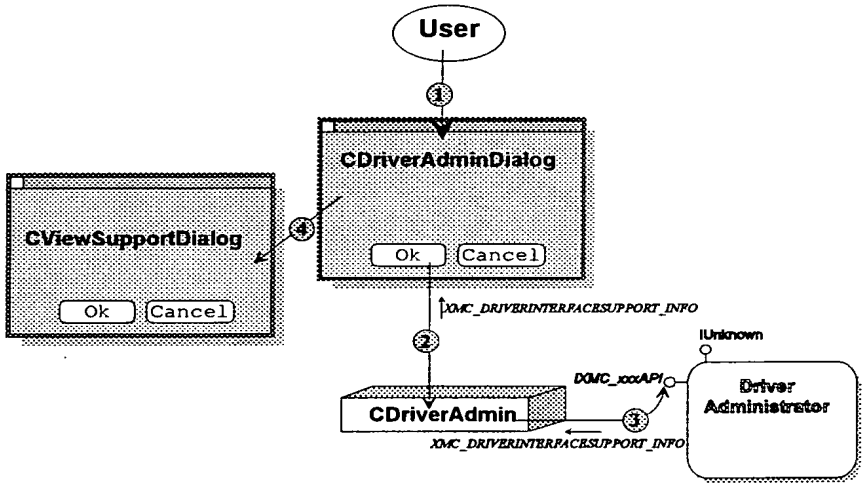
~~Figure 5~~ *Adding a Driver.*

FIG. 55



~~Figure 6~~ *Removing a Driver.*

FIG. 56



~~Figure 7~~ *View Support.*

FIG. 57

filed w/Exhibits A-H

HUGHES, MULTER & SCHACHT, P.S.

PATENT APPLICATION

DOCKET NO. P2966

UTILITY X COUNTRY _____ DIVISION _____

DESIGN ☐ CLIENT _____ CIP _____

REISSUE ☐ PCT _____ CONTINUATION _____

APPLICANT Dave Brown

TITLE Motion control system

SERIAL NO. 08/454,736 ART UNIT _____ EXAMINER _____ FILED 5/30, 1995

PRIORITY _____

FOREIGN FILINGS PCT filed 5/30/96 (P3158a) claiming priority

ORIGIN of 08/454,736 filed 5/30/95

OFFICE ACTIONS		RESPONSES	
FMPA dated	7/12/95	Issue fee paid	7/1/97
FMPA due	8/12/95	filed a CIP	5/30/96
FMPA done	7/25/95	(P3158)	
IDS done	8/21/95		
Supp IDS done	9/25/96		
O.A.	10/2/96		
Response due	1/2/97		
Response done	1/2/97		
Allowed	4/11/97		
Drwgs + Issue fee due	7/11/97		
Drwgs done	6/30/97		

ISSUE FEE DUE 7/11/97 ISSUE FEE PAID 7/1/97

ALLOWED 4/11/97 PATENT NO. 5,691,897 DATE 11/25, 1997 EXPIRES _____

ASSIGNMENT

DATED 6/28/95

ASSIGNOR Brown + Clark

ASSIGNEE Roy-G-Biv Corp

ADDRESS _____

INTEREST _____

RECORDED 7/31/95 REEL 7580 FRAME 0835

U.S. ANNUITIES

	DUE	PAID
4TH YEAR	5/25/01	5/23/01
8TH YEAR	5/25/05	5/14/05
12TH YEAR	5/25/09	

APPLICANT Brown, Dave

SERIAL NO. 08/454,736 ATTORNEY MRS

PATENT NO. 5,691,897

CLIENT _____ COUNTRY _____

DOCKET NO. P2966

MOTION CONTROL SYSTEM (original Patent)

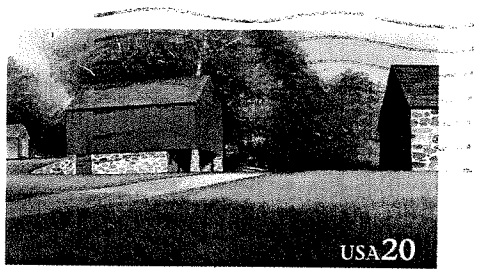
L. J. COWIE CO., INC. ROCKVILLE, MARYLAND – (301) 424-4950

PLEASE DATE STAMP AND RETURN TO ACKNOWLEDGE RECEIPT
IN THE U.S. PATENT & TRADEMARK OFFICE IN RE:

NEW INCOMPLETE PATENT APPLICATION
Applicants: DAVID W. BROWN and JAY S. CLARK
Docket No. P2966
Title: MOTION CONTROL SYSTEMS
Enc: Specification and Claims (97 pages, 24 claims, Appendices A-H)
Forty-six (46) sheets informal drawings
Unsigned Declaration and Power of Attorney
Unsigned Small Entity Status Form
--Indep. Inventor
Letter requesting treatment under 37 CFR 1.53
Transmittal Letter in duplicate
Letter of Express Mail No. EG150680122US
Check #4132 for $409 filing fee
MRS:gjn Mailed: May 30, 1995

08/454,735

RECEIVED
JUL 31 1995
HUGHES, MULTER & SCHACHT, P.S.



HUGHES, MULTER & SCHACHT
1720 Iowa Street
Bellingham, WA 98226

DOCKETED
8/30/95 fa 2198
8/30/95 fa 2199

DOCKETED
5/30/96 ffl 2200
2/28/96 ffl 2201
7/30/96 fa 2202

08/454736

LAW OFFICES OF
HUGHES, MULTER & SCHACHT, P.S.
A PROFESSIONAL SERVICE CORPORATION

ROBERT B. HUGHES
RICHARD D. MULTER
MICHAEL R. SCHACHT

14711 N.E. 29TH PLACE
SUITE 245
BELLEVUE, WA 98007-7666
(206) 453-5701
FAX: (206) 881-5878

1720 IOWA STREET
BELLINGHAM, WA 98226-4702
(360) 647-1296 (BHM.)
(360) 988-2061 (SUMAS)
(206) 447-9172 (SEA.)
FAX: (360) 671-2489

PATENT, TRADEMARK,
COPYRIGHT LAW
& LITIGATION

REPLY TO
BELLINGHAM
OFFICE

May 30, 1995

PATENT

Commissioner of Patents and Trademarks
U.S. Patent & Trademark Office
Washington, D.C. 20231

SUBJECT: New U.S. Patent Application
Inventors: DAVID W. BROWN and JAY S. CLARK
Docket No.: P2966
Title: MOTION CONTROL SYSTEMS
Express Mail Label EG150680122US
Date of Deposit: May 30, 1995

Dear Sir:

The captioned application, a copy of which is attached, is being filed pursuant to the provisions of 37 CFR 1.53(b), (d); the nature of the incompletion being the unavailability of the inventors to execute the accompanying declaration. In accordance with the revisions of 37 CFR 1.10, we ask that this application be accorded an effective filing date of even date herewith notwithstanding the fact. We look forward to return receipt, in due course, of the Patent Office notification of incompletion, at which time we will submit the completed declaration of the inventors.

Respectfully submitted,

Michael R. Schacht

Michael R. Schacht, Reg. No. 33,550
Hughes, Multer & Schacht, P.S.
1720 Iowa Street
Bellingham, WA 98226
(360) 647-1296
Fax: (360) 671-2489

08/454736

LAW OFFICES OF
HUGHES, MULTER & SCHACHT, P.S.
A PROFESSIONAL SERVICE CORPORATION

ROBERT B. HUGHES
RICHARD D. MULTER
MICHAEL R. SCHACHT

14711 N.E. 29TH PLACE
SUITE 245
BELLEVUE, WA 98007-7666
(206) 453-5701
FAX: (206) 881-5878

1720 IOWA STREET
BELLINGHAM, WA 98226-4702
(360) 647-1296 (BHM.)
(360) 988-2061 (SUMAS)
(206) 447-9172 (SEA.)
FAX: (360) 671-2489

PATENT, TRADEMARK, COPYRIGHT LAW & LITIGATION

REPLY TO
BELLINGHAM
OFFICE

May 30, 1995

PATENT

Commissioner of Patents and Trademarks
U.S. Patent and Trademark Office
Washington, D.C. 20231

Sir:

Transmitted herewith for filing is the incomplete patent application in re:

Inventors: DAVID W. BROWN and JAY S. CLARK

For: MOTION CONTROL SYSTEMS

Attorneys' Docket No.: P2966

Date of Deposit: May 30, 1995

"Express Mail" mailing label number: EG150680122US

I hereby certify that this application is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, United States Patent and Trademark Office, Washington, D.C. 20231.

Gloria J. Nims
Secretary

PLEASE GIVE THIS APPLICATION THE FILING DATE OF MAY 30, 1995.

08/454736

Commissioner of Patents and Trademarks
U.S. Patent and Trademark Office
Washington, D.C. 20231

May 30, 1995
Attorneys' Docket No. P2966
Express Mail Label No. EG150680122US

Sir:

Transmitted herewith for filing is an incomplete patent application in re:

Applicants: DAVID W. BROWN and JAY S. CLARK

Title: MOTION CONTROL SYSTEMS

1. Enclosed are:

   a) Specification, claims, abstract, and appendices (85 pgs. of description, 11 pgs. of claims, 1 pg. abstract, and Appendices A-H;

   b) Forty-six (46) sheets of informal drawings;

   c) Unexecuted combined Declaration and Power of Attorney;

   d) Unexecuted declaration claiming Small Entity Status--Independent Inventor;

   e) Letter requesting treatment under 37 CFR 1.10, Express Mail Filing Date, Label No. EG150680122US;

   f) Letter requesting treatment under 37 CFR 1.53(b), (d);

   g) A check in the amount of $409 for Filing Fee; and

   h) A stamped return receipt postcard.

2. The filing fee has been calculated as shown below:

	Col. 1 Number Filed	Col. 2 Number Extra	SMALL ENTITY Rate	Fee
			Basic Fee	$365
Total Claims	24	20 = 4	X 11=	44
Indep. Claims	3	3 = 0	X 38=	
____ Multiple Dependent Claim Presented			+120=	
			TOTAL	**$409**

3. The Commissioner is hereby authorized to charge payment of the following fees associated with this communication and during the pendency of this application, or credit any overpayment to Account No. 08-3260. A duplicate copy of this sheet is enclosed.

   X Any additional filing fees under 37 CFR 1.16 for the presentation of extra claims.
   X Any patent application processing fees under 37 CFR 1.17.

Michael R. Schacht

Michael R. Schacht, Reg. No. 33,550
HUGHES, MULTER & SCHACHT, P.S.
1720 Iowa St., Bellingham, WA 98226
(360) 647-1296
Fax: (360) 671-2489

UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. 20231
www.uspto.gov

APPLICATION NUMBER	FILING DATE	GRP ART UNIT	FIL FEE REC'D	ATTY.DOCKET.NO	DRAWINGS	TOT CLAIMS	IND CLAIMS
10/021,669	12/10/2001	2121	370	P213976	64	10	1

CONFIRMATION NO. 5760

FILING RECEIPT

OC000000007301057

Michael R. Schacht
2801 Meridian St., Suite 202
Bellingham, WA 98225-2412

Date Mailed: 01/11/2002

Receipt is acknowledged of this nonprovisional Patent Application. It will be considered in its order and you will be notified as to the results of the examination. Be sure to provide the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION when inquiring about this application. Fees transmitted by check or draft are subject to collection. Please verify the accuracy of the data presented on this receipt. **If an error is noted on this Filing Receipt, please write to the Office of Initial Patent Examination's Customer Service Center. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the USPTO processes the reply to the Notice, the USPTO will generate another Filing Receipt incorporating the requested corrections (if appropriate).**

Applicant(s)

David W. Brown, Bingen, WA;
Jay S. Clark, Bingen, WA;

Domestic Priority data as claimed by applicant

THIS APPLICATION IS A CON OF 09/191,981 11/13/1998
WHICH IS A CON OF 08/656,421 05/30/1996 PAT 5,867,385
WHICH IS A CIP OF 08/454,736 05/30/1995 PAT 5,691,897

Foreign Applications

If Required, Foreign Filing License Granted 01/11/2002

Projected Publication Date: Request for Non-Publication Acknowledged

Non-Publication Request: Yes

Early Publication Request: No

**** SMALL ENTITY ****

Title

Motion control systems

Preliminary Class

700

LICENSE FOR FOREIGN FILING UNDER
Title 35, United States Code, Section 184
Title 37, Code of Federal Regulations, 5.11 & 5.15

GRANTED

The applicant has been granted a license under 35 U.S.C. 184, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" followed by a date appears on this form. Such licenses are issued in all applications where the conditions for issuance of a license have been met, regardless of whether or not a license may be required as set forth in 37 CFR 5.15. The scope and limitations of this license are set forth in 37 CFR 5.15(a) unless an earlier license has been issued under 37 CFR 5.15(b). The license is subject to revocation upon written notification. The date indicated is the effective date of the license, unless an earlier license of similar scope has been granted under 37 CFR 5.13 or 5.14.

This license is to be retained by the licensee and may be used at any time on or after the effective date thereof unless it is revoked. This license is automatically transferred to any related applications(s) filed under 37 CFR 1.53(d). This license is not retroactive.

The grant of a license does not in any way lessen the responsibility of a licensee for the security of the subject matter as imposed by any Government contract or the provisions of existing laws relating to espionage and the national security or the export of technical data. Licensees should apprise themselves of current regulations especially with respect to certain countries, of other agencies, particularly the Office of Defense Trade Controls, Department of State (with respect to Arms, Munitions and Implements of War (22 CFR 121-128)); the Office of Export Administration, Department of Commerce (15 CFR 370.10 (j)); the Office of Foreign Assets Control, Department of Treasury (31 CFR Parts 500+) and the Department of Energy.

NOT GRANTED

No license under 35 U.S.C. 184 has been granted at this time, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" DOES NOT appear on this form. Applicant may still petition for a license under 37 CFR 5.12, if a license is desired before the expiration of 6 months from the filing date of the application. If 6 months has lapsed from the filing date of this application and the licensee has not received any indication of a secrecy order under 35 U.S.C. 181, the licensee may foreign file the application pursuant to 37 CFR 5.15(b).

05-205 A/A

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
08/454,736	05/30/95	BROWN	D 0796

02/2/0712

MICHAEL R. SCHACHT
HUGHES MULTER & SCHACHT
1720 IOWA STREET
BELLINGHAM WA 98226

DATE MAILED: 0000

NOTICE TO FILE MISSING PARTS OF APPLICATION FILING DATE GRANTED

07/12/95

An Application Number and Filing Date have been assigned to this application. However, the items indicated below are missing. The required items and fees identified below must be timely submitted **ALONG WITH THE PAYMENT OF A SURCHARGE** for items 1 and 3-6 only of $130.00 for large entities or $65.00 for small entities who have filed a verified statement claiming such status. The surcharge is set forth in 37 CFR 1.16(e).

If all required items on this form are filed within the period set below, the total amount owed by applicant as a ☒ large entity, ☐ small entity (verified statement filed), is $130.00.

Applicant is given **ONE MONTH FROM THE DATE OF THIS LETTER, OR TWO MONTHS FROM THE FILING DATE** of this application, **WHICHEVER IS LATER,** within which to file all required items and pay any fees required above to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

1. ☐ The statutory basic filing fee is: ☐ missing ☐ insufficient. Applicant as a ☐ large entity ☐ small entity, must submit $_____ to complete the basic filing fee.

2. ☐ Additional claim fees of $_____ as a ☐ large entity, ☐ small entity, including any required multiple dependent claim fee, are required. Applicant must submit the additional claim fees or cancel the additional claims for which fees are due.

3. ☐ The oath or declaration:
   ☐ is missing.
   ☐ does not cover the newly submitted items.

   An oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date is required.

4. ☐ The oath or declaration does not identify the application to which it applies. An oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.

5. ☒ The signature(s) to the oath or declaration is/are: ☒ missing; ☐ by a person other than the inventor or a person qualified under 37 CFR 1.42, 1.43, or 1.47. A properly signed oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.

6. ☐ The signature of the following joint inventor(s) is missing from the oath or declaration:

   _____ An oath or declaration listing the names of all inventors and signed by the omitted inventor(s), identifying this application by the above Application Number and Filing Date, is required.

7. ☐ The application was filed in a language other than English. Applicant must file a verified English translation of the application and a fee of $_____ under 37 CFR 1.17(k), unless this fee has already been paid.

8. ☐ A $_____ processing fee is required since your check was returned without payment. (37 CFR 1.21(m)).

9. ☐ Your filing receipt was mailed in error because your check was returned without payment.

10. ☐ The application does not comply with the Sequence Rules. See attached Notice to Comply with Sequence Rules 37 CFR 1.821-1.825.

320 MM 08/14/95 0845473e
1 205 65.00 CK

11. ☐ Other.

Direct the response to Box Missing Part and refer any questions to the Customer Service Center at (703) 308-1202.

A copy of this notice MUST be returned with the response.

#3

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	BROWN ET AL.	) Docket No.:) P2966
Serial No.:	08/454,736	)
Filed:	05/30/95	) Application) Branch
Title:	MOTION CONTROL SYSTEMS	)

SUBMISSION OF LATE DECLARATION AND SURCHARGE

Commissioner of Patents and Trademarks
U.S. Patent And Trademark Office
Washington, D.C. 20231

ATTENTION: APPLICATION PROCESSING DIVISION
SPECIAL PROCESSING & CORRESPONDENCE BRANCH

Sir:

In response to the "NOTICE TO FILE MISSING PARTS OF APPLICATION - FILING DATE GRANTED" mailed July 12, 1995, (copy enclosed) and stating that the Oath or Declaration to the above-identified application was missing, enclosed herewith is a Combined Declaration and Power of Attorney, Verified Statements Claiming Small Entity Status -- Independent Inventor and Small Business Concern. A check for the $65.00 surcharge fee is enclosed. It is believed that no other fee is due at this time to maintain this application in full force and effect, however, if any such fee is due, please charge it to Deposit Account No. 08-3260.

140

Signed at Bellingham, County of Whatcom, State of Washington, this 25th day of July, 1995.

Respectfully submitted,

BROWN ET AL.

By: Michael R. Schacht

Michael R. Schacht, Reg. No. 33,550
HUGHES, MULTER & SCHACHT, P.S.
1720 Iowa Street
Bellingham, WA 98226
(360) 647-1296

CERTIFICATE OF MAILING (37 CFR 1.8a)

I hereby certify that this paper (along with any paper referred to as being attached or enclosed) is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner of Patents and Trademarks, Washington, D.C. 20231, on date shown below.

July 25, 1995
Date

GLORIA J. NIMS
(Print name of person mailing paper)

Gloria J. Nims
(Signature of person mailing paper)

PATENT

Attorney's Docket No. P2966

Applicant or Patentee: DAVID W. BROWN and JAY S. CLARK

Serial or Patent No.: 08/ 454,736

Filed or Issued: May 30, 1995

For: MOTION CONTROL SYSTEMS

VERIFIED STATEMENT (DECLARATION) CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) and 1.27(b))—INDEPENDENT INVENTOR

As a below named inventor, I hereby declare that I qualify as an independent inventor as defined in 37 CFR 1.9(c) for purposes of paying reduced fees under Section 41(a) and (b) of Title 35, United States Code, to the Patent and Trademark Office with regard to the invention entitled MOTION CONTROL SYSTEMS

described in

- ☐ the specification filed herewith.
- ☒ application serial no. 08/ 454,736, filed May 30, 1995.
- ☐ patent no. _____, issued _____.

I have not assigned, granted, conveyed or licensed and am under no obligation under contract or law to assign, grant, convey or license, any rights in the invention to any person who could not be classified as an independent inventor under 37 CFR 1.9(c) if that person had made the invention, or to any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a nonprofit organization under 37 CFR 1.9(e).

Each person, concern or organization to which I have assigned, granted, conveyed, or licensed or am under an obligation under contract or law to assign, grant, convey, or license any rights in the invention is listed below:

- ☐ no such person, concern, or organization
- ☒ persons, concerns or organizations listed below*

*NOTE: *Separate verified statements are required from each named person, concern or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27).*

FULL NAME ROY-G-BIV Corporation

ADDRESS 150 East Jewett Blvd.
White Salmon, WA 98672

☐ INDIVIDUAL ☒ SMALL BUSINESS CONCERN ☐ NONPROFIT ORGANIZATION

FULL NAME _____

ADDRESS _____

☐ INDIVIDUAL ☐ SMALL BUSINESS CONCERN ☐ NONPROFIT ORGANIZATION

FULL NAME _____

ADDRESS _____

☐ INDIVIDUAL ☐ SMALL BUSINESS CONCERN ☐ NONPROFIT ORGANIZATION

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of pay-

ing, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b)).

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issuing thereon, or any patent to which this verified statement is directed.

DAVID W. BROWN

Name of inventor

Date 6/28/95

Signature of inventor

JAY S. CLARK

Name of inventor

Date 6/28/95

Signature of inventor

Name of inventor

Date

Signature of inventor

PATENT

#3

Attorney's Docket No. P2966

COMBINED DECLARATION AND POWER OF ATTORNEY

(ORIGINAL, DESIGN, NATIONAL STAGE OF PCT, SUPPLEMENTAL, DIVISIONAL, CONTINUATION OR CIP)

As a below named inventor, I hereby declare that:

TYPE OF DECLARATION

This declaration is of the following type: (*check one applicable item below*)

- [X] original
- [ ] design
- [ ] supplemental

NOTE: *If the declaration is for an International Application being filed as a divisional, continuation or continuation-in-part application do not check next item; check appropriate one of last three items.*

- [ ] national stage of PCT

NOTE: *If one of the following 3 items apply then complete and also attach ADDED PAGES FOR DIVISIONAL, CONTINUATION OR CIP.*

- [ ] divisional
- [ ] continuation
- [ ] continuation-in-part (CIP)

INVENTORSHIP IDENTIFICATION

WARNING: *If the inventors are each not the inventors of all the claims an explanation of the facts, including the ownership of all the claims at the time the last claimed invention was made, should be submitted.*

My residence, post office address and citizenship are as stated below next to my name, I believe I am the original, first and sole inventor (*if only one name is listed below*) or an original, first and joint inventor (*if plural names are listed below*) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

TITLE OF INVENTION

MOTION CONTROL SYSTEMS

SPECIFICATION IDENTIFICATION

the specification of which: (*complete (a), (b) or (c)*)

(a) [ ] is attached hereto.

(b) [X] was filed on May 30, 1995 as [X] Serial No. 08/454,736
or [ ] Express Mail No., *as Serial No. not yet known* _____
and was amended on _____ (*if applicable*).

NOTE: *Amendments filed after the original papers are deposited with the PTO which contain new matter are not accorded a filing date by being referred to in the declaration. Accordingly, the amendments involved are those filed with the application papers or, in the case of a supplemental declaration, are those amendments claiming matter not encompassed in the original statement of invention or claims. See 37 CFR 1.67.*

(c) ☐ was described and claimed in PCT International Application No. _____ filed on _____ and as amended under PCT Article 19 on _____ (*if any*).

ACKNOWLEDGEMENT OF REVIEW OF PAPERS AND DUTY OF CANDOR

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, § 1.56(a).

☐ In compliance with this duty there is attached an information disclosure statement. 37 CFR 1.97.

PRIORITY CLAIM

I hereby claim foreign priority benefits under Title 35, United States Code, § 119 of any foreign application(s) for patent or inventor's certificate or of any PCT international application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application(s) for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) of which priority is claimed.

(complete (d) or (e))

(d) ☒ no such applications have been filed.

(e) ☐ such applications have been filed as follows.

NOTE: Where item (c) is entered above and the International Application which designated the U.S. claimed priority check item (e), enter the details below and make the priority claim.

EARLIEST FOREIGN APPLICATION(S), IF ANY FILED WITHIN 12 MONTHS (6 MONTHS FOR DESIGN) PRIOR TO THIS U.S. APPLICATION

COUNTRY	APPLICATION NUMBER	DATE OF FILING (day, **month**, year)	PRIORITY CLAIMED UNDER 37 USC 119
			☐ YES NO ☐
			☐ YES NO ☐
			☐ YES NO ☐
			☐ YES NO ☐
			☐ YES NO ☐

ALL FOREIGN APPLICATION(S), IF ANY FILED MORE THAN 12 MONTHS (6 MONTHS FOR DESIGN) PRIOR TO THIS U.S. APPLICATION

POWER OF ATTORNEY

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. (*List name and registration number*)

3 Robert B. Hughes (19,304); Richard D. Multer (20,661); and Michael R. Schacht, Reg. No. 33,550.

(check the following item, if applicable)

☐ Attached as part of this declaration and power of attorney is the authorization of the above-named attorney(s) to accept and follow instructions from my representative(s).

SEND CORRESPONDENCE TO

MICHAEL R. SCHACHT
HUGHES, MULTER & SCHACHT, P.S.
1720 IOWA STREET
BELLINGHAM, WA 98226

DIRECT TELEPHONE CALLS TO:
(Name and telephone number)

MICHAEL R. SCHACHT
(206) 647-1296
FAX: (206) 671-2489

DECLARATION

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

SIGNATURE(S)

Full name of **sole or first inventor** DAVID W. BROWN 1-00
Inventor's signature
Date 6/28/95 Country of Citizenship U.S.A.
Residence White Salmon, Washington Wa
Post Office Address P.O. Box 1278, 150 East Jewett Blvd.
White Salmon, WA 98672

Full name of **second joint inventor, if any** JAY S. CLARK 2-00
Inventor's signature
Date 6/28/95 Country of Citizenship U.S.A.
Residence Seattle, Washington Wa
Post Office Address 557 Roy Street, Suite 175
Seattle, WA 98109

CHECK PROPER BOX(ES) FOR ANY OF THE FOLLOWING ADDED PAGE(S) WHICH FORM A PART OF THIS DECLARATION

☐ Signature for third and subsequent joint inventors. *Number of pages added* _____

☐ Signature by administrator(trix), executor(trix) or legal representative for deceased or incapacitated inventor. *Number of pages added* _____

☐ Signature for inventor who refuses to sign or cannot be reached by person authorized under 37 CFR 1.47. *Number of pages added* _____.

• • •

☐ Added pages to combined declaration and power of attorney for divisional, continuation, or continuation-in-part (CIP) application.

☐ Number of pages added _____

• • •

☐ Authorization of attorney(s) to accept and follow instructions from representative

• • •

If no further pages form a part of this Declaration then end this Declaration with this page and check the following item

☒ **This declaration ends with this page**

PATENT

Attorney's Docket No. P2966

Applicant or Patentee: DAVID W. BROWN and JAY S. CLARK

Serial or Patent No.: 08/ 454,736

Filed or Issued: May 30, 1995

For: MOTION CONTROL SYSTEMS

VERIFIED STATEMENT (DECLARATION) CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) and 1.27(c))—SMALL BUSINESS CONCERN

I hereby declare that I am

- ☐ the owner of the small business concern identified below:
- ☒ an official of the small business concern empowered to act on behalf of the concern identified below:

NAME OF CONCERN ROY-G-BIV Corporation

ADDRESS OF CONCERN 150 East Jewett Blvd.
White Salmon, WA 98672

I hereby declare that the above identified small business concern qualifies as a small business concern as defined in 13 CFR 121.3-18, and reproduced in 37 CFR 1.9(d), for purposes of paying reduced fees under Section 41(a) and (b) of Title 35, United States Code, in that the number of employees of the concern, including those of its affiliates, does not exceed 500 persons. For purposes of this statement, (1) the number of employees of the business concern is the average over the previous fiscal year of the concern of the persons employed on a full-time, part-time or temporary basis during each of the pay periods of the fiscal year, and (2) concerns are affiliates of each other when either, directly or indirectly, one concern controls or has the power to control the other, or a third-party or parties controls or has the power to control both.

I hereby declare that rights under contract or law have been conveyed, to and remain with the small business concern identified above with regard to the invention, entitled

MOTION CONTROL SYSTEMS

by inventor(s) DAVID W. BROWN and JAY S. CLARK

described in

- ☐ the specification filed herewith.
- ☒ application serial no. 08/ 454,736, filed May 30, 1995.
- ☐ patent no. _____, issued _____.

If the rights held by the above identified small business concern are not exclusive, each individual, concern or organization having rights in the invention is listed below* and no rights to the invention are held by any person, other than the inventor, who would not qualify as an independent inventor under 37 CFR 1.9(c) if that person made the invention, or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a nonprofit organization under 37 CFR 1.9(e).

*NOTE: *Separate verified statements are required from each named person, concern or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27).*

NAME _____

ADDRESS _____

☐ INDIVIDUAL ☐ SMALL BUSINESS CONCERN ☐ NONPROFIT ORGANIZATION

NAME _____

ADDRESS _____

☐ INDIVIDUAL ☐ SMALL BUSINESS CONCERN ☐ NONPROFIT ORGANIZATION

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small business entity is no longer appropriate. (37 CFR 1.28(b)).

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issuing thereon, or any patent to which this verified statement is directed.

NAME OF PERSON SIGNING David W. Brown

TITLE OF PERSON OTHER THAN OWNER President and CEO

ADDRESS OF PERSON SIGNING 150 East Jewett Blvd.
White Salmon, WA 98672

SIGNATURE _____ Date 6/28/95

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTY. DOCKET NO./TITLE
08/454,736	05/30/95	BROWN D	P2966

0252/0712

MICHAEL R. SCHACHT
HUGHES MULTER & SCHACHT
1720 IOWA STREET
BELLINGHAM WA 98226

DATE MAILED: 0000

NOTICE TO FILE MISSING PARTS OF APPLICATION FILING DATE GRANTED

07/12/95

An Application Number and Filing Date have been assigned to this application. However, the items indicated below are missing. The required items and fees identified below must be timely submitted **ALONG WITH THE PAYMENT OF A SURCHARGE** for items 1 and 3-6 only of $130.00 for large entities or $65.00 for small entities who have filed a verified statement claiming such status. The surcharge is set forth in 37 CFR 1.16(e).

If all required items on this form are filed within the period set below, the total amount owed by applicant as a X large entity, ☐ small entity (verified statement filed), is $130.00.

Applicant is given **ONE MONTH FROM THE DATE OF THIS LETTER, OR TWO MONTHS FROM THE FILING DATE** of this application, **WHICHEVER IS LATER,** within which to file all required items and pay any fees required above to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

1. ☐ The statutory basic filing fee is: ☐ missing ☐ insufficient. Applicant as a ☐ large entity ☐ small entity, must submit $_____ to complete the basic filing fee.

2. ☐ Additional claim fees of $_____ as a ☐ large entity, ☐ small entity, including any required multiple dependent claim fee, are required. Applicant must submit the additional claim fees or cancel the additional claims for which fees are due.

3. ☐ The oath or declaration:
   ☐ is missing.
   ☐ does not cover the newly submitted items.

   An oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date is required.

4. ☐ The oath or declaration does not identify the application to which it applies. An oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.

5. X ☐ The signature(s) to the oath or declaration is/are: X missing; ☐ by a person other than the inventor or a person qualified under 37 CFR 1.42, 1.43, or 1.47. A properly signed oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.

6. ☐ The signature of the following joint inventor(s) is missing from the oath or declaration:

   _____ An oath or declaration listing the names of all inventors and signed by the omitted inventor(s), identifying this application by the above Application Number and Filing Date, is required.

7. ☐ The application was filed in a language other than English. Applicant must file a verified English translation of the application and a fee of $_____ under 37 CFR 1.17(k), unless this fee has already been paid.

8. ☐ A $_____ processing fee is required since your check was returned without payment. (37 CFR 1.21(m)).

9. ☐ Your filing receipt was mailed in error because your check was returned without payment.

10. ☐ The application does not comply with the Sequence Rules. See attached Notice to Comply with Sequence Rules 37 CFR 1.821-1.825.

11. ☐ Other.

Direct the response to Box Missing Part and refer any questions to the Customer Service Center at (703) 308-1202.

A copy of this notice MUST be returned with the response.

OFFICE COPY

UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. 20231
www.uspto.gov

APPLICATION NUMBER	FILING DATE	GRP ART UNIT	FIL FEE REC'D	ATTY.DOCKET.NO	DRAWINGS	TOT CLAIMS	IND CLAIMS
10/021,669	12/10/2001	2121	435	P213976	64	10	1

CONFIRMATION NO. 5760

UPDATED FILING RECEIPT

OC000000007543706

Michael R. Schacht
2801 Meridian St., Suite 202
Bellingham, WA 98225-2412

Date Mailed: 02/27/2002

Receipt is acknowledged of this nonprovisional Patent Application. It will be considered in its order and you will be notified as to the results of the examination. Be sure to provide the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION when inquiring about this application. Fees transmitted by check or draft are subject to collection. Please verify the accuracy of the data presented on this receipt. **If an error is noted on this Filing Receipt, please write to the Office of Initial Patent Examination's Customer Service Center. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the USPTO processes the reply to the Notice, the USPTO will generate another Filing Receipt incorporating the requested corrections (if appropriate).**

Applicant(s)

David W. Brown, Bingen, WA;
Jay S. Clark, Bingen, WA;

Domestic Priority data as claimed by applicant

THIS APPLICATION IS A CON OF 09/191,981 11/13/1998
WHICH IS A CON OF 08/656,421 05/30/1996 PAT 5,867,385
WHICH IS A CIP OF 08/454,736 05/30/1995 PAT 5,691,897

Foreign Applications

if Required, Foreign Filing License Granted 01/11/2002

Projected Publication Date: Request for Non-Publication Acknowledged

Non-Publication Request: Yes

Early Publication Request: No

**** SMALL ENTITY ****

Title

Motion control systems

Preliminary Class

151

700

LICENSE FOR FOREIGN FILING UNDER
Title 35, United States Code, Section 184
Title 37, Code of Federal Regulations, 5.11 & 5.15

GRANTED

The applicant has been granted a license under 35 U.S.C. 184, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" followed by a date appears on this form. Such licenses are issued in all applications where the conditions for issuance of a license have been met, regardless of whether or not a license may be required as set forth in 37 CFR 5.15. The scope and limitations of this license are set forth in 37 CFR 5.15(a) unless an earlier license has been issued under 37 CFR 5.15(b). The license is subject to revocation upon written notification. The date indicated is the effective date of the license, unless an earlier license of similar scope has been granted under 37 CFR 5.13 or 5.14.

This license is to be retained by the licensee and may be used at any time on or after the effective date thereof unless it is revoked. This license is automatically transferred to any related applications(s) filed under 37 CFR 1.53(d). This license is not retroactive.

The grant of a license does not in any way lessen the responsibility of a licensee for the security of the subject matter as imposed by any Government contract or the provisions of existing laws relating to espionage and the national security or the export of technical data. Licensees should apprise themselves of current regulations especially with respect to certain countries, of other agencies, particularly the Office of Defense Trade Controls, Department of State (with respect to Arms, Munitions and Implements of War (22 CFR 121-128)); the Office of Export Administration, Department of Commerce (15 CFR 370.10 (j)); the Office of Foreign Assets Control, Department of Treasury (31 CFR Parts 500+) and the Department of Energy.

NOT GRANTED

No license under 35 U.S.C. 184 has been granted at this time, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" DOES NOT appear on this form. Applicant may still petition for a license under 37 CFR 5.12, if a license is desired before the expiration of 6 months from the filing date of the application. If 6 months has lapsed from the filing date of this application and the licensee has not received any indication of a secrecy order under 35 U.S.C. 181, the licensee may foreign file the application pursuant to 37 CFR 5.15(b).