MVS Dynamic Reconfiguration Management

by R. Cwiakala
J. D. Haggar
H. M. Yudenfriend

This paper presents an overview of the Dynamic Reconfiguration Management (DRM) function of MVS/ESA™ and its support of the IBM Enterprise System/9000™ family of machines. Dynamic Reconfiguration Management is the ability to select a new I/O configuration definition without needing to perform a power-on reset (POR) of the hardware or an initial program load (IPL) of the MVS operating system. Dynamic Reconfiguration Management allows the installation to add, delete, or modify definitions for channel paths, control units, and I/O devices, in both the software and hardware I/O configurations.

Introduction

Information systems perform a critical function within business enterprises. For many businesses, operation twenty-four hours a day, seven days a week is essential. In international operations, time zone differences are often an important factor. Outage and duration of outage can easily be correlated to loss of productivity and/or to lost revenue. The Dynamic Reconfiguration Management (DRM) function [1] has been designed to support the objective of continuous system operation in the following ways:

- Increasing system availability by allowing changes to the I/O configuration while systems are in productive operation.
- Eliminating the disruption caused by power-on reset and initial program load, as well as the subsequent restarting of subsystems and networks.

This function complements the nondisruptive installation capability of Enterprise Systems Connection (ESCON™) control units allowed by fiber optic technology and connection topology, although DRM also supports existing control units and devices.

Large IBM mainframe processors and the control programs that operate on them both require definitions of the system I/O configuration in order to effectively support the execution of application-initiated I/O operations. Since the advent of the IBM 3081 processor, the definition of processor (hardware) I/O configuration has been performed through the I/O Configuration Program (IOCP), and the definition of the software I/O configuration has been performed either through the system generation (SYSGEN) process or the MVS configuration program (MVSCP) [2].

In the IBM Multiple Virtual Storage (MVS) operating system, the processes, though separate, can use common card-image input, which includes information required by both the hardware and the software I/O configuration definition. With separate processes, however, there is still the potential for certain mismatch errors between the two configuration definitions. Such errors may not be detected until an attempt to initiate an I/O operation fails on an I/O device. The recovery may require a rerun of both MVSCP and IOCP and a subsequent power-on reset (POR) of the hardware and initial program load (IPL) of the operating system.

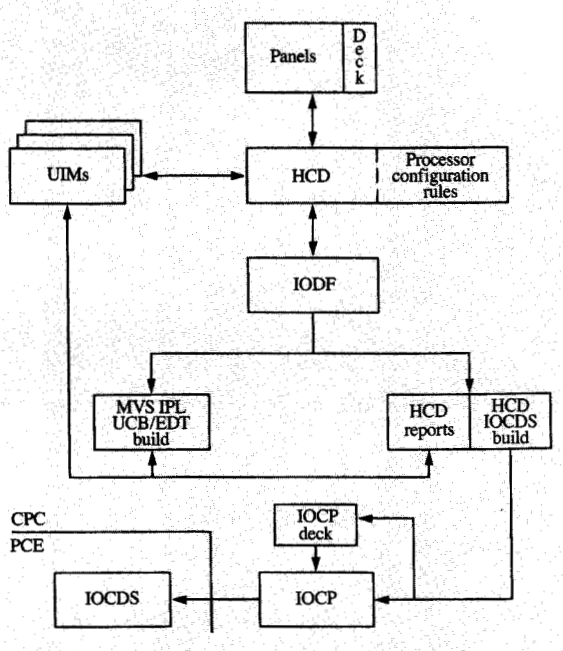

Figure 1

Interactive I/O definition process details.

In 1990, an improvement to MVS called the Hardware Configuration Definition™ (HCD) [3] function was introduced. Its objective was to consolidate the hardware and software I/O configuration definition processes under a single interactive end-user interface, and to address the problem of late detection of inconsistencies between the hardware and software definitions. HCD provides an interactive panel-driven capability that supports both the hardware and software I/O configuration definition functions, as shown in **Figure 1**. HCD validates all input against hardware and software "rules" and detects inconsistencies and errors, allowing an interactive user to make immediate corrections. Even with HCD, changes to the current definition still require a POR and IPL.

Dynamic Reconfiguration Management (DRM), introduced with System/390® (and supported by MVS/ESA™ SP4.2.0), builds on the HCD function. With DRM, an HCD-created I/O definition file (IODF) can be used to change the I/O configuration definition without requiring the outage previously associated with the process (i.e., without a POR and IPL). At control program initialization, "architected" (formally defined) interfaces between the control program and the hardware allow the control program to determine whether its representation of the I/O configuration is consistent with that defined for the hardware. Once consistency has been verified, the current I/O configuration definition can be updated with a new definition by use of an HCD interactive panel, or by an MVS operator command that invokes the DRM function. The control program determines the changes required to the existing definition (i.e., additions, deletions, and modifications) and makes the necessary changes to the software and, through the architected interfaces, to the hardware. Changes are synchronized with existing I/O activity to minimize or eliminate disruption. The resultant hardware definition can be written to a hardware I/O configuration data set (IOCDS) for use during subsequent initialization.

The control program provides services that allow application programs to be notified of a planned or completed configuration change. This is vital to applications that are sensitive to the I/O configuration, such as the ESCON Manager™.

The DRM function is supported both in basic mode and logically partitioned (LPAR) mode. This function, invoked in a logical partition running MVS, changes the hardware definition of I/O resources across all affected partitions. It permits a single point of control for DRM related to hardware definitions. To ensure that deleted resources do not adversely affect currently executing applications within other partitions, installations must coordinate the planned I/O configuration change across affected partitions before making the change.

Background information

MVS builds the following data areas to represent the software I/O configuration definition:

- *Unit control blocks (UCBs)* A UCB represents the software definition of a device to MVS. Each device type is represented to the software by a unit information module (UIM) which is included in the product that contains the device support. The UIM provides device-dependent UCB information and indicates whether or not the device type supports the dynamic capability. During IPL, MVS builds a UCB for each device in the configuration definition.
- *Eligible device table (EDT)* The EDT is an installation-defined and -named representation of the devices that are eligible for allocation. The EDT also defines the relationships among these devices. During IPL, MVS builds an EDT representing the EDT definition.

Dynamic Reconfiguration Management design

Introducing Dynamic Reconfiguration Management into a software and hardware structure designed around static I/O configuration definitions provided a unique set of challenges.

Many of the software data structures dealing with I/O configuration definitions can trace their origins back to System/360™. Those that intersected with hardware function (e.g., performance data gathering) had to reside in contiguous real storage that was established when the control program was initialized. The I/O configuration definition (i.e., UCBs, EDT, and other related data structures) was prevalidated and loaded at control program initialization. Any changes to the I/O configuration had to be made within the context of this loaded definition. The hardware I/O configuration definition had many of the same properties. As with software, definitions were prevalidated and loaded into contiguous main storage at hardware initialization (i.e., POR). Neither hardware nor software could handle additions or modifications to the current configuration definition.

Performance implications required the continued use of contiguous real storage. The design had to ensure that enough contiguous real storage would be allocated at hardware and software initialization to accommodate dynamic reconfiguration. Prevalidated configuration definitions were chosen over more granular definitions (i.e., single-element definitions), because of practical hardware and software implementation limits. The number of channel paths, control units, or devices that are supported by any hardware or software implementation is finite. With the more granular approach, a check against implementation limits would have to be done during the activation of a change. In addition to the performance implication, finding out that a planned change exceeded implementation limits at activation time instead of at definition time was unacceptable.

Although prevalidated configuration definitions were chosen, it was clear that deleting an entire configuration definition and adding a new one was unacceptable if the incremental change involved only one or a few elements. For this reason, the mechanism chosen was to make any incremental change against the current definition. Any determination of incremental change required that the hardware and the software have a consistent interpretation of the current configuration definition. This was necessary in order to guarantee a valid set of changes to be applied to the current definition. The design had to account for this mechanism, and for the interfaces for communicating this consistent interpretation of the configuration definition between hardware and software.

Before the introduction of DRM, the "current" hardware I/O configuration definition was the one loaded at POR. The default hardware I/O configuration definition, for a subsequent POR, was the one loaded at the last POR. Since DRM allows the hardware I/O configuration definition to change without a POR, the question of default had to be addressed. The proper default should be the configuration represented in the hardware system area (HSA), but should it become the default as soon as the dynamic change is made, or after the installation has done some testing to ensure adequate function and performance? The design needed a means for determining when the hardware would be allowed to accept a new default, and which IOCDS included the representation of the default. Also, the hardware had to be able to determine whether the new default did in fact match the configuration defined in the HSA.

In addition to those design considerations which affected the control program software and hardware, there were many that had to be addressed between applications sensitive to configuration changes and the control program. Installations had to be able to evolve to the new environment of DRM with little or no disruption. Unprepared applications had to be shielded from the loss of resources (e.g., I/O devices they were using) and from the appearance or disappearance of configuration data they were not prepared to handle. For them, the world of dynamic configuration required a static appearance.

Conversely, applications that were prepared to handle dynamic change needed to have control over their destiny. Mechanisms had to be created that allowed applications to accommodate resource deletions at times that did not conflict with the use of those resources, and to be told when the I/O configuration environment had changed so that they could adapt (use new resource, change their view of the environment they were monitoring, etc.). The design had to define an appropriate set of services that were downward compatible, but also be able to exploit the new dynamic environment. Similarly, control program structures needed to be put in place to shield those applications that had not yet evolved to the new environment.

The remainder of this paper addresses the design solutions to these problems.

Software operations

Over the life of MVS, applications, supplier products, IBM products, and MVS components have built up dependencies on the static data structures that represent the I/O configuration definition within MVS. Dynamic modifications to these data structures were precluded because there was no way to guarantee the integrity of the system and the consistency of the data. The first challenge of providing a dynamic I/O configuration capability was to provide an upward-compatible way to achieve dynamic changes to these data structures while allowing old programs to continue to run without being affected by the dynamic changes. Additionally, it was required that all changes be performed in a nondisruptive manner and be transparent to the user.

Another design goal of the dynamic I/O configuration function was to maintain the separation of the

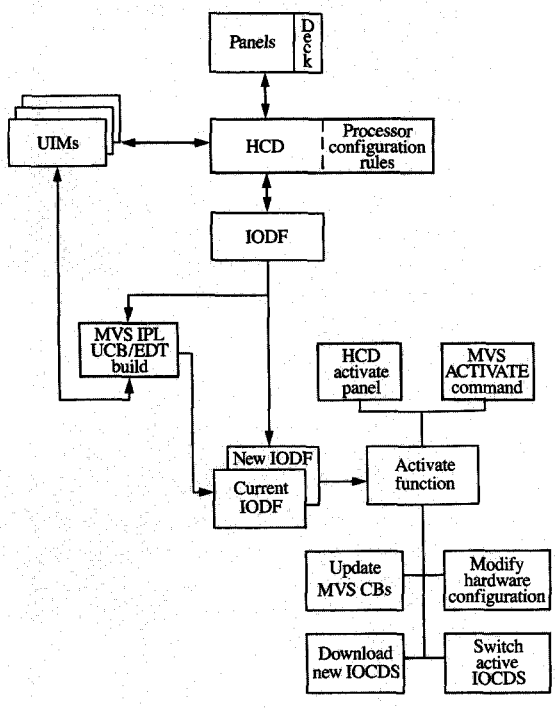

Figure 2

Configuration change process. The configuration change process is invoked either by the CMS ACTIVATE command or through the HCD activate panel. A new I/O configuration definition is achieved by the installation selecting a target IODF and activating it. MVS compares the active I/O configuration definition to the target I/O configuration definition and makes the changes.

configuration definition task from the configuration activation task. The former task encompasses everything from defining the topology of the I/O configuration (i.e., which control units attach to which channel paths, which devices attach to which control units) to defining the usage characteristics of the I/O components. It also includes consideration of physical cabling constraints and performance goals for the I/O components. This task is typically performed by highly skilled systems programmers, well before the time when the actual configuration change occurs.

The configuration activation process actually puts the machine into the state that reflects the new I/O configuration. It is typically performed by operations personnel during scheduled outages (weekends and holidays).

Another goal of the dynamic I/O configuration function was to reduce, if not totally eliminate, cases in which the software I/O configuration definition does not correspond to the hardware I/O configuration definition.

- *Configuration activation*

Configuration activation is shown schematically in **Figure 2**. The installation selects a new I/O configuration definition by specifying a new hardware or software definition, or both, within the active IODF, or by identifying a new IODF (see **Figure 3**). The definitions are then compared in order to derive the set of changes that must be implemented.

After determining the set of changes, the control program validates that the changes are permissible. For example, devices that are being deleted must be off-line to the system and not in use (see the section on UCB pinning). While performing this validity checking, the system prevents any logical reconfigurations from taking place. This prevents a device, device path, or channel path from being reconfigured on-line after MVS verifies that it can be deleted.

After verifying the changes, MVS issues a signal so that all programs that need to monitor I/O configuration changes are given a chance to prepare for the configuration change. At this point MVS verifies that all UCBs that are being deleted are not in use by the system. After this verification, MVS makes all the changes to the hardware definition, makes the new software data structures visible to the rest of the operating system, and issues a final signal to inform programs that the configuration change is complete.

If any step of the configuration change process fails, MVS returns the configuration to its original state. All planned changes must be completed or none are made. This ensures that the hardware configuration definition contained in the channel subsystem corresponds to the hardware configuration definition in the software files (IODFs).

Basing the MVS Dynamic Reconfiguration Management process on predefined IODFs provides the following features:

- The configuration definition task continues to be separated from the configuration activation task.
- Target configuration definitions are guaranteed to be consistent with the machine implementation limits.
- Recovery to an existing predefined configuration state is simplified. One alternative to having predefined configuration definitions would be to have an individual (or a program) enter configuration change commands one at a time to the control program. If a system outage occurred while the configuration was being changed, the state of the hardware configuration definition would be unpredictable. The MVS compare function provides a consistent set of changes to be performed between any

```
                    Activate New Hardware and Software Configuration

Specify or revise the values for the IODF activation.

Currently active IODF    : SYS1.IODF01
 Processor ID            : CEC1      Processor description
 Configuration ID        : TSOAQ     Configuration description
 EDT ID                  : 01        EDT description

IODF to be activated     : SYS1.IODF02
  Processor ID             CEC1____  +
  Configuration ID         TSOAQ___  +
  EDT ID                   01        +

Test only:                                  _____   (Yes or No)
Allow hardware deletes / FORCE option       _____   (Yes or No)

Write IOCDS                                 _____   (Yes or No)
Switch IOCDS name for next POR              _____   (Yes or No)

Command ===>

F1=Help   F2=Split   F4=Prompt   F5=Reset   F9=Swap   F12=Cancel
```

Figure 3

HCD Activate Panel used to initiate a dynamic reconfiguration change.

two configuration definitions contained in IODFs. This guaranteed consistency is the cornerstone in the MVS recovery process in case a system outage occurs while a configuration change is in progress (see the section on recovery).

- The operator interface is simplified. Prespecified, complete I/O configuration definitions avoid requiring the operator to manually enter configuration definitions in real time, write a separate program or CLIST, or maintain additional sources of configuration change data that must be applied to a given configuration definition.
- Any-to-any configuration changes are permitted. Since the process compares two I/O configuration definitions and determines the set of changes required, in real time, the installation is not required to track the history of how the current configuration was derived. Only those changes that the control program needs to make are performed. Operational procedures are not required to determine the individual steps necessary to achieve the desired configuration change. For example, operations personnel do not need to be aware of the individual devices being added. However, if activating a configuration definition requires that I/O resources be deleted or modified, operator involvement is needed to ensure that the I/O components that are affected are not in use. Typically this requires taking the device logically off-line so that it cannot be allocated for use by applications.

- *Defining I/O devices*

Programs written for earlier versions of MVS are unprepared to cope with dynamic configuration changes. To maintain compatibility with existing programs, and to permit configuration-related data structures to become dynamic, the following concepts were implemented:

- A device type either supports or does not support the dynamic capability. Existing products which provide

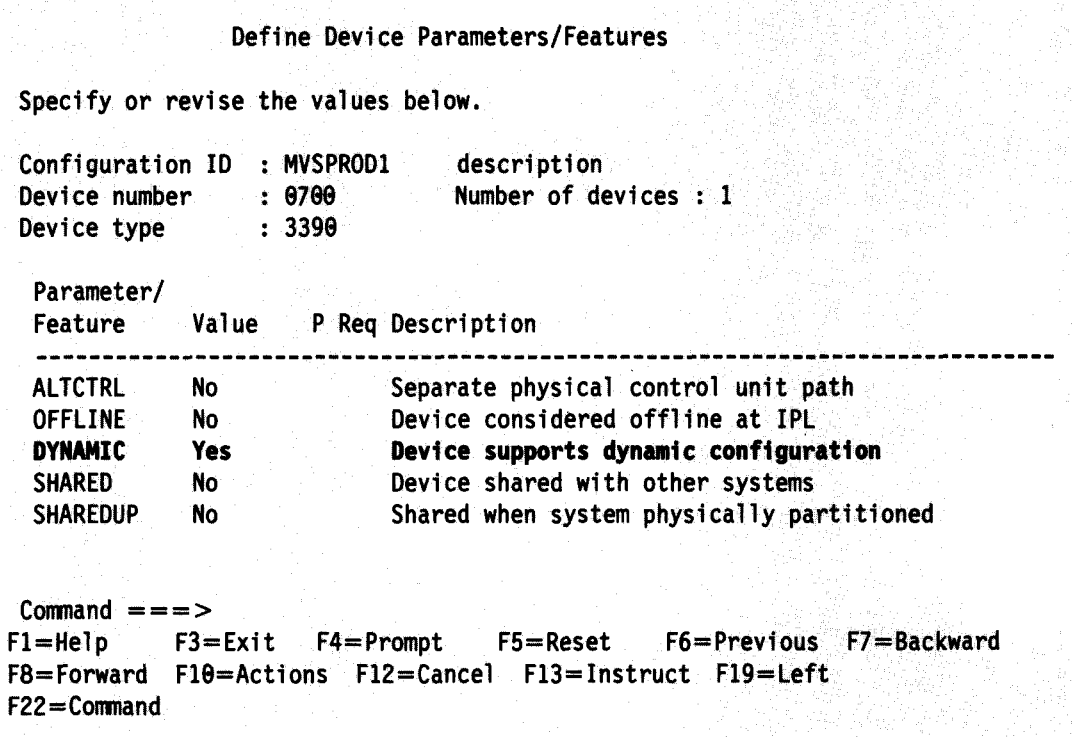

Figure 4

Definition of device parameters. If the device type supports dynamic reconfiguration, this panel allows the customer to specify whether the device is to be defined as installation-static or dynamic.

support for given device types maintain static-device-dependent data structures and are unprepared to handle the dynamic addition and deletion of UCBs.

- A device whose device type supports the dynamic capability may be defined by the installation as dynamic or not dynamic (**Figure 4**). Many existing programs, including customer programs, supplier programs, and IBM products, depend on device-related data structures such as UCBs and the EDT, or use existing MVS programming services which access these data structures, and are unprepared to handle dynamic changes to these structures.

These new concepts resulted in three software categories of device definitions:

1. *Static*—the device support code does not support the dynamic capability.
2. *Installation-static*—the device type supports the dynamic capability, but the installation specifies in the device definition that the device is not to be treated as dynamic.
3. *Dynamic*—the device type supports the dynamic capability and the installation specifies that the device is to be treated as dynamic.

- *New MVS services*

MVS now provides new programming services to obtain UCB and EDT information in a dynamic environment. Existing lookup and scanning services and existing data structures are limited to those devices defined as static and installation-static.

Because UCBs which are defined as dynamic are accessible only to programs which use the new programming services, unchanged programs which use existing UCB services are shielded from encountering dynamic UCBs which are subject to deletion.

The new MVS programming services are the following:

- UCBPIN—pins or unpins a UCB.
  - PIN—prevents a UCB from being dynamically deleted.
  - UNPIN—releases the pin on the UCB.
- UCBLOOK—obtains the UCB address for a given device number.
- UCBSCAN—scans through UCBs (returns UCB address or copy of UCB data).
- EDTINFO—obtains information from the EDT.
- IOCINFO—obtains current MVS I/O configuration token.

- *UCB pinning*

The pinning concept allows authorized programs to mark a UCB as ineligible for deletion until the program performs a corresponding unpin. Programs which access only the UCBs of allocated devices do not need to perform pinning, because an allocated UCB cannot be dynamically deleted.

When the configuration definition of a device is dynamically deleted, MVS dynamically deletes the UCB representing the device and any associated device-related control blocks. Also, when the configuration definition of a device is dynamically modified, MVS dynamically deletes the UCB representing the device and dynamically adds a new UCB, and its device-related data structures, representing the new device definition. Thus, pinning prevents both the deletion and the modification of device definitions.

MVS deletes UCBs only for devices which are not in use by the system. To be a candidate for deletion, a device must be both off-line and unallocated. However, there are many programs which use off-line or unallocated devices, or which use UCBs for devices without regard for whether the device is on-line or off-line. These programs need a method of preventing a UCB from being deleted. Additionally, environmental restrictions make MVS device allocation impractical for many operating system programs to use as a way to prevent the deletion of a device.

- *MVS I/O configuration token*

Dynamic changes to the UCBs and the EDT introduce the following problems:

- Programs which need to obtain information about the current I/O configuration (e.g., by scanning the set of UCBs representing the configuration) may encounter inconsistent results if the set of UCBs representing the I/O configuration is changing.
- Programs which are sensitive to the relationship between a device number and a UCB may encounter inconsistent results if a device definition is dynamically modified from one device type to another.
- Programs which maintain lists of UCB addresses, and then validate a given UCB address before using it, may encounter inconsistent results if a UCB is dynamically deleted and another UCB is later dynamically added at the same storage address.
- Programs sensitive to the logical grouping of devices for the purpose of allocation (i.e., the contents of the EDT) may encounter inconsistent results.

The same problems are introduced when a dynamic device reconfiguration (DDR) swap occurs which exchanges the contents of two UCBs. A DDR swap changes the relationship between the device number and the UCB for the devices involved in the swap. The DDR swap problem exists independently of DRM.

To solve these problems, the concept of an MVS I/O configuration token has been introduced. This token uniquely represents the state of the I/O configuration to MVS.

Using the MVS I/O configuration token with MVS services

The MVS I/O configuration token can be used in conjunction with the new MVS programming services (UCBSCAN, UCBLOOK, UCBPIN, EDTINFO) to ensure that the information returned by the services is consistent with the configuration definition represented by the token.

MVS sets the initial MVS I/O configuration token during IPL. The MVS I/O configuration token is updated when a new I/O configuration definition is activated, or when a DDR swap occurs. An I/O configuration token consists of multiple subtokens representing all aspects of the I/O configuration that are accessible via programming services. For a given dynamic configuration change, only the affected portions of the token are updated. Programming services ensure consistency with respect to the data being returned. For example, if a dynamic configuration change only updates the EDT, programming services which return UCB information do not indicate that anything is inconsistent.

The MVS I/O configuration token provides a way for programs that require knowledge of configuration data to detect that dynamic I/O configuration changes have occurred. For example, a program can determine whether previously obtained configuration information has become obsolete. Specifically, the token allows a program to detect changes in the set of devices (useful for a program which scans the set of UCBs), in a specific device definition, or in the EDT definition. The token is an optional parameter for programs using the new MVS services. The following examples demonstrate how programs can use the token with specific MVS services.

- *UCBSCAN*

The UCBSCAN service allows a program to scan the set of UCBs by repetitively invoking the service. The

service requires an input work area which is used to keep state information across repetitive invocations of the service.

It is possible that the configuration can change between invocations of the UCBSCAN service. By storing the current configuration token into the work area on its initial scan, the UCBSCAN service can detect whether a configuration change has occurred on a subsequent call, and notify the caller via a return code. The caller can then choose to restart the scan.

- *UCBLOOK*

The UCBLOOK service allows an authorized program to obtain a UCB address for a given device number. This service also provides a PIN option which ensures that the UCB address returned has been pinned.

The MVS I/O configuration token concept allows a program to maintain a list of UCB addresses or device numbers without keeping all UCBs in the list pinned, provided that the program keeps a configuration token with the list. MVS ensures that the device definition for the input device number or UCB address has not changed since the time at which the input configuration token was created. If the device definition has changed, the service sets a return code to indicate that the device definition is inconsistent with the token.

The configuration token allows programs to detect the following cases, which otherwise would not be detected:

1. The configuration definition for a particular device number has changed (because the service validates both that a UCB exists for the device number and that the device definition is consistent with the token).
2. A UCB is deleted, but a different UCB is later added at the same storage address (because the service validates both that the UCB address represents a valid UCB and that the device definition is consistent with the token).

- *EDTINFO*

The EDTINFO service allows the caller to obtain information about the current EDT. The configuration token allows the caller to ensure that multiple invocations of the service return consistent data.

- *Operations on software device definitions*

The following sections summarize the operations that can be performed on the software definitions of devices.

Dynamic devices

The capability is provided to add, delete, or modify the software definition of a dynamic device. This causes MVS to dynamically add and delete UCBs.

The capability is also provided to dynamically redefine a dynamic device as installation-static. This is useful in making the device and its UCB available to programs which use the pre-dynamic programming services for access to UCBs.

Installation-static devices

The ability is provided to dynamically add the software definition of an installation-static device. This causes MVS to dynamically add a UCB and make it available to programs which use pre-dynamic services to access UCBs.

It is not possible to dynamically delete or modify the software definition of an installation-static device, because there are no existing programming primitives or serialization techniques that allow the removal of UCBs from the system without causing incompatibilities with existing applications, supplier and IBM products, and MVS operating system code. Additionally, dynamically adding paths to or deleting them from an installation-static device is not allowed because of the incompatibilities this would cause with existing MVS code. This capability allows an installation to exploit DRM without any changes to its applications, although it is limited to adding only new device definitions.

The ability is provided to dynamically redefine an installation-static device as dynamic. This causes MVS to invalidate the installation-static UCB and build a new UCB for the dynamic device (i.e., the UCB address is changed for the device). Therefore, there is a risk that programs which have the address of the old UCB might attempt to use it and encounter problems. For this reason, the system storage that contains an installation-static UCB is never freed, but rather is functionally invalidated (marked in such a way that it is not usable). This avoids errors that could result in storage overlays.

This capability allows the user to begin exploiting the full dynamic capability (deletes and modifies, in addition to adds) as soon as the installation has converted all its old programs to exploit the new UCB services and to follow the new serialization requirements for dynamic devices. If this capability were not provided, the installation would have to re-IPL the system in order to start exploiting the dynamic I/O capability.

[*Note*: This transition is allowed only if the sole difference between the source definition of the device and the target definition of the device is that the source definition is installation-static and the target definition is dynamic. No other information about the device definition, relative to the operating system (as opposed to the hardware), can be different (e.g., device type). This rule is imposed to minimize the probability that activating the wrong IODF will cause work or valuable system storage to be wasted.]

Static devices

No capability is provided to dynamically add, delete, or modify the software definition of a static device. However,

activating an IODF which includes the definition of a new static device is allowed. In this case, the activation is accepted, but the UCB is not dynamically added. If a hardware definition for the device is requested, it is added. A subsequent IPL of the same IODF makes the UCB available.

This capability minimizes the number of IODFs that must be created by the installation and therefore simplifies the management of these files. For example, if the installation wanted to add both dynamic devices and static devices to the configuration, it need only create one IODF with both changes. Only when the installation needs to actually start using the static devices must it re-IPL the system. Since the hardware definition is added at activation time, no POR of the machine is required.

Hardware operations

To achieve the design goals of MVS Dynamic Reconfiguration Management, three types of hardware functions were required. First, the software required a mechanism to determine whether or not the actual configuration definition contained in the hardware matched the I/O configuration definition contained in the IODF. This mechanism was required so that the software could determine that the changes to be applied to the current hardware I/O configuration definition were derived from comparing the target IODF to a current IODF which contained a representation of the current hardware configuration. If the starting definition could not be guaranteed, the incremental change that was determined by comparing the source and target IODFs might not be valid, and performing these changes would yield unpredictable results.

Next, the software required a set of hardware primitives to allow the system to modify the I/O configuration definition contained in the HSA (tables and control storage used by the channel subsystem). These primitives include the ability to nondisruptively modify the definitions for channel paths, control units, and I/O devices. The LPAR environment also introduced the requirement to include in this set of primitives a mechanism to ensure that only one partition at a time makes dynamic changes to the hardware configuration definition.

Finally, a set of machine-dependent functions was required in order to let the installation control how much system resource to set aside for dynamic growth, whether dynamic changes are to be allowed, and which partitions are allowed to make them.

• *Hardware/software synchronization*

In order for the DRM process to predictably update the hardware configuration definition, the software must have complete knowledge of the hardware I/O configuration definition. This is achieved by creating a hardware configuration token that uniquely identifies a hardware I/O configuration definition. This hardware configuration token is kept within the IODF that is used to create the initial hardware definition of the machine. When the machine is initially powered on (POR), this initial hardware configuration token is loaded into the HSA. Any time the installation attempts to update the hardware configuration definition by activating a new hardware configuration definition, MVS retrieves the hardware configuration token from the HSA and verifies that the currently active IODF contains a matching hardware configuration token. If it does not, the hardware and software I/O configuration definitions are said to be out of synchronization, and no dynamic hardware changes are allowed. If the currently active IODF does contain a hardware configuration token that matches the one contained in the HSA, the hardware and software I/O configuration definitions are said to be synchronized, and both hardware and software dynamic I/O configuration changes are allowed.

The installation can optionally make configuration changes that affect only the software definition. This option is available whether or not the software and hardware I/O configuration definitions are synchronized. Additionally, this capability can be used to restore synchronization as needed. The section on CPC-wide changes in LPAR describes the conditions under which this capability is required.

Figure 5 shows the data flows for initially setting the hardware configuration token.

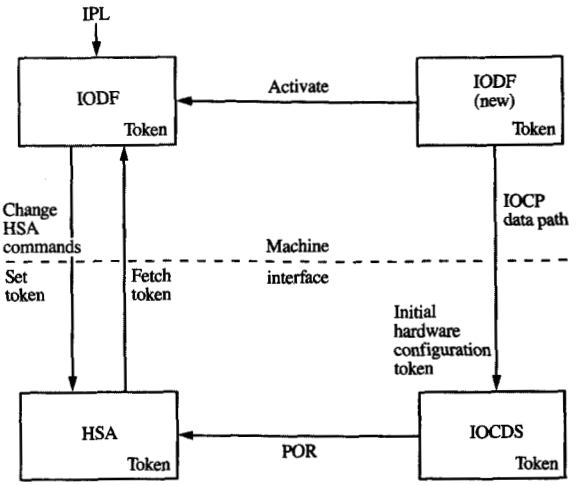

Figure 5

Synchronization of configurations. The active IODF, the new IODF, the HSA, and the IOCDS must all be kept synchronized.

• *Automatic selection of the initial IODF*
A problem that frequently occurs in data processing installations is that a software configuration definition used for IPL is incompatible with the hardware configuration definition that was used for POR of the machine. When these two definitions are not intended for use with each other, unpredictable errors may occur, including inability of the operating system to initialize, or even worse, initializing and causing errors to occur that may jeopardize data integrity.

In addition to its usage to determine whether or not the software and hardware I/O configuration definitions are synchronized (for the purpose of allowing or prohibiting dynamic hardware changes), the hardware configuration token can be used by MVS to automatically select an IODF that matches the hardware configuration definition. At the option of the installation, MVS searches the designated initialization volume for the first IODF that contains a hardware configuration token matching the current hardware configuration token in the HSA. By carefully choosing the naming conventions for its IODFs, the installation can control which IODF is selected, in case multiple IODFs contain the same hardware configuration definition. The MVS search for the matching IODF starts with an IODF data set suffix of '00' and proceeds sequentially to 'FF' (hex values). By using this automatic synchronization capability, the installation can minimize the chances of initializing the system with an incorrect I/O configuration definition.

• *Operations on I/O components*
Enterprise System/9000™ (ES/9000™) processors provide an interface that allows the control program to manipulate the hardware I/O configuration definition and to query I/O configuration state information. This interface was designed with the following objectives:

- *Machine independence* The interface to manipulate the hardware configuration definition is implementation-independent. Specific machine implementations are free to construct their internal representations of the configuration in any manner they choose.
- *Efficiency* The architected primitives that manipulate the internal hardware structures may take relatively long periods of time to complete (i.e., several milliseconds), depending on the nature and scope of the change and the system activity at the time of the change. Therefore, these primitives were designed to complete asynchronously to the program.
- *Minimal disruption* Only those I/O components that are affected by the configuration change are momentarily suspended while internal machine structures are updated. New I/O operations presented to the channel subsystem may be temporarily prevented from starting, and disconnected operations may be prevented from completing. These effects are transparent to the user.
- *Recoverability across a set of planned changes* Mechanisms are provided that allow the program to maintain state information relative to a set of planned configuration changes. This state information persists across IPLs of the software and is visible across logical partitions. This capability allows the operating system to continue configuration changes from the point at which a failure occurred.

• *Operations on channel path definitions*
Operations on channel paths allow the installation to dynamically redefine a channel path. This allows the installation to change the definition of its I/O channel paths among various types of devices. For example, a channel path that is currently defined as a block-multiplexor channel path can by dynamically redefined as a byte-multiplexor channel path and switched (or recabled) to a new set of devices. Additionally, ESCON-capable channel paths can be dynamically redefined to be ESCON channel paths, ESCON conversion channel paths, or ESCON CTC channel paths. Finally, if a configuration definition is not correct (e.g., an ESCON channel path is incorrectly defined to be a block channel path), the hardware configuration definition can be dynamically changed to a definition that is usable by the hardware without having to POR the machine.

• *Operations on control unit definitions*
Most aspects of the control unit definitions are dynamically modifiable by the system. Control unit definitions can be dynamically added, deleted, and modified. The modification of a control unit definition includes the capability to add channel paths to it, remove channel paths from it, change the unit address ranges that it is defined to recognize, etc.

• *Operations on I/O device definitions*
Most aspects of I/O device definitions are dynamically modifiable. Device definitions can be dynamically added, deleted, and modified. The modification of a device definition can include adding and deleting control units to which it may be attached, setting a preferred path for I/O operations, and turning either or both the status detection facility and the interface timeout function on or off.

• *Changing the default IOCDS for the next POR*
The default hardware I/O configuration data set (IOCDS) for the next POR of the machine is the IOCDS that was used to POR the current I/O configuration. Since DRM provides the ability to change the existing I/O configuration definition in the hardware, customers also require the

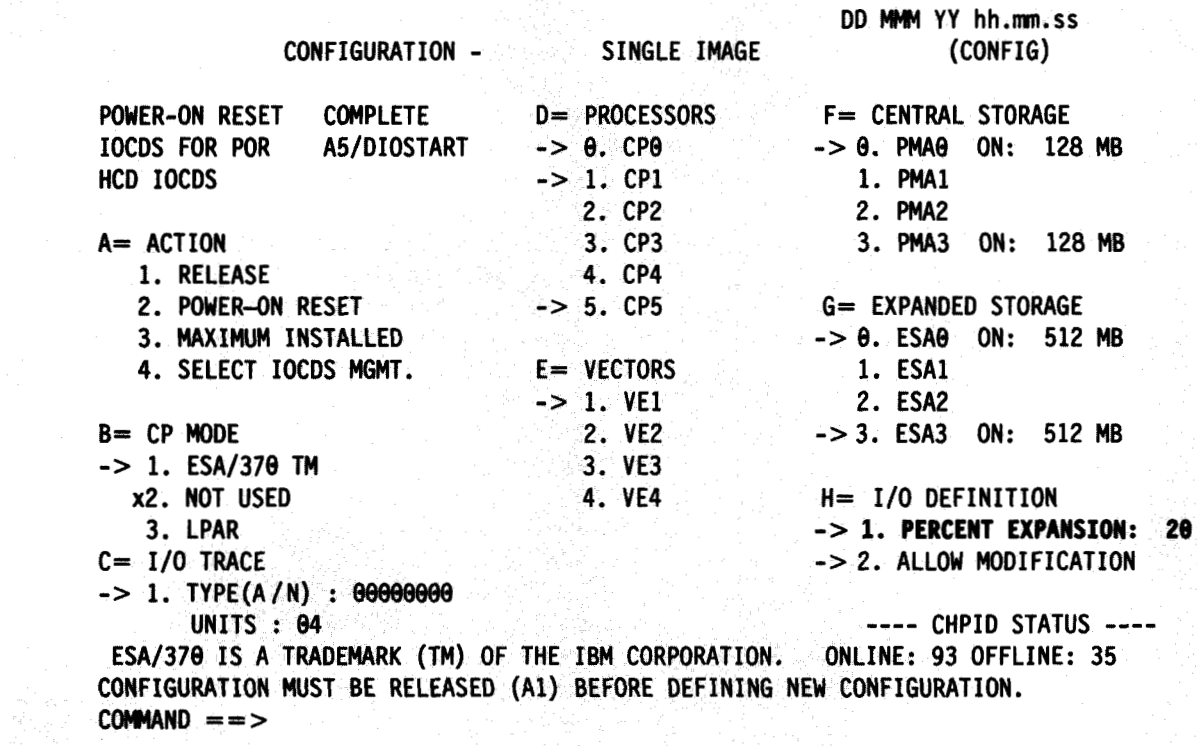

Figure 6

CONFIG frame. The service processor for the ES/9000 processors provides the ability to control whether or not dynamic configuration changes are to be permitted, as well as the amount of storage to be set aside in the HSA for growth of the configuration.

ability to change the default IOCDS that is used for the next POR (see Figure 3). It must be possible to specify this capability

- At the time a dynamic configuration change is attempted, so that an installation can continue to run with the new I/O configuration definition even after an outage forces it to POR the machine.
- After a dynamic configuration has completed, in case the installation should be run with the new I/O configuration for a period of time to test that it is correct, before committing to it for the next POR.

The switch-active IOCDS function works only if the target IOCDS contains an I/O configuration definition that matches the I/O configuration definition contained in the HSA. This check is performed by comparing the hardware configuration token contained in the channel subsystem with the hardware configuration token contained in the target IOCDS.

- *Space requirements*

During the POR process, the installation specifies how much system resource (i.e., storage) is to be set aside for growing the I/O configuration definition. The interface for specifying this growth is in terms of a percentage of the I/O configuration definition specified in the POR IOCDS (see **Figure 6**). Space for the channel, control unit, and I/O device specifications is increased by the specified percentage. This *expansion factor* was chosen as the external interface for designated system growth requirements for the following reasons:

- *Simplicity* A percentage growth factor is a simple interface for the customer to understand. An alternative approach that was dismissed was to have the customer

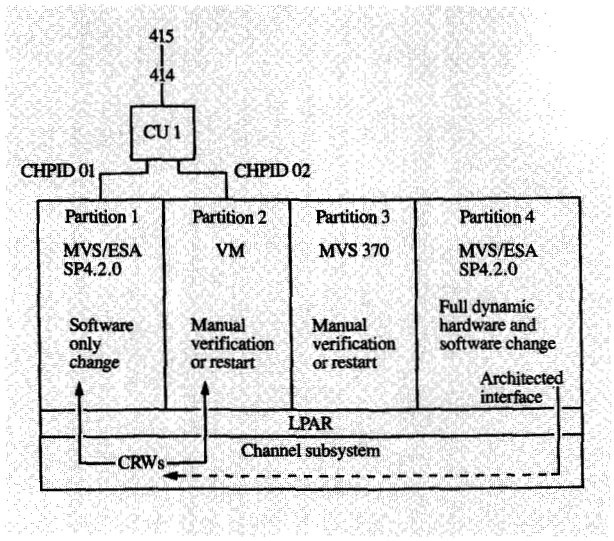

Figure 7

CPC-wide changes in LPAR. Coordinating dynamic I/O configuration changes in an LPAR environment requires that the customer perform software-only changes in the first $n-1$ partitions, and the full hardware and software change last, in the nth partition.

specify the number of additional I/O components intended to be added. This implied detailed planning for the expected growth in terms of specific numbers of devices and control units, and how they connected.

- *Machine model independence* A percentage growth factor can be translated by the machine into the appropriate amount of storage, which can vary with future implementations.

- *CPC-wide changes in LPAR*

The hardware primitives defined to allow the software to modify the I/O configuration in HSA were intended to operate on the I/O configuration definition of the entire machine (i.e., across logical partitions), so that an MVS operating system could perform changes to other partitions (which might be running other operating systems). Thus, these other partitions could be re-IPLed in order to change only the software definition, as the hardware definition has been changed for them. Optionally, these other operating systems could support a subset of the DRM function that would enable them to modify their own software structures without having to implement the functions that alter the hardware definition of the machine.

Cross-LPAR serialization

As was the case with the software data structures, machine internals have evolved over time. These internal control structures have also evolved from essentially static designs, and they cannot easily be updated by multiple logical partitions at the same time. For these reasons, a new serialization primitive was defined that allows only one logical partition at a time to make dynamic changes. This new serialization primitive puts the logical partition into configuration mode. In this mode, the hardware primitives that modify the hardware I/O configuration definition are allowed to be executed.

The primitive designed to perform this serialization atomically compares the hardware configuration token passed by the software with the hardware configuration token contained in the HSA. If the token in the HSA is valid and matches the token passed by the software, configuration mode is established, and no other partitions can enter configuration mode. When the software has completed its changes to the hardware configuration definition, it leaves configuration mode and sets the new hardware I/O configuration token into the HSA.

Program notification of configuration changes across LPARs

The ability of DRM to manipulate the entire hardware I/O configuration definition of the processor from a central point of control introduces the requirement that the affected logical partitions must be notified of the configuration change when additional I/O components become visible to the program or when I/O components that were visible are no longer visible. This function was accomplished by the existing IBM Enterprise Systems architecture for installed-parameters-initialized (IPI) channel report words (CRWs). However, the existing mechanism only provided for the addition and removal of hardware definitions; it did not cover the modification of definitions. A new CRW had to be defined that allowed a more granular notification of I/O configuration changes. This notification is provided with an installed-parameters-modified (IPM) CRW. Upon receipt of this CRW, the operating system determines the scope of the configuration changes and updates its internal state description to represent the new hardware configuration definition. **Figure 7** shows a dynamic change in an LPAR environment.

MVS software-only configuration changes

The CPC-wide capability afforded by DRM allows the installation to use MVS to maintain the I/O configuration on the entire machine, not just the MVS partition initiating the configuration change. If the configuration change removes any definitions from the configuration or modifies any definitions, care must be taken not to adversely affect work in other partitions that may be using the affected I/O resources. The MVS partition initiating the configuration change will never delete an I/O resource that is in use within its own partition. However, there is no way to verify the status of the I/O resource in the other partitions.

DRM provides the ability for the installation to coordinate dynamic configuration changes across multiple MVS/ESA systems that support DRM by providing a software-only change capability. This capability allows MVS to verify that the I/O resources that will be deleted by the configuration change are not in use by the partition. Performing the software-only configuration change in the first $n - 1$ partitions before doing the complete configuration change in the nth partition ensures that none of the I/O resources being deleted are in use. When the final configuration change is done, the hardware changes are made and the other partitions are notified using the CRWs described above.

The software-only capability is also useful for other environments, e.g., MVS running as a virtual guest on a VM system or on a processor that does not support the DRM function. The operating system can still perform configuration changes to its I/O configuration definition, independently of the ability to change the hardware definition. This is useful if an MVS customer would like to change the initial EDT. Finally, software-only changes give an installation the ability to reacquire hardware–software synchronization if that becomes necessary, e.g., because of an IPL using the wrong IODF.

Recovery

The primary motivation for Dynamic Reconfiguration Management was to move toward continuous availability for the customer. It was essential that system failures that may occur during a dynamic I/O configuration change not require a POR of the machine to recover to a known configuration definition state (i.e., the source or target configuration definition). To achieve this objective, DRM provides the necessary hardware and software mechanisms to continue dynamic I/O configuration changes across IPLs of the software, or in a different LPAR that may have initiated the dynamic change.

The MVS design, which only allows the activation of predefined IODFs, serves two purposes for recovery:

1. The configuration data that initiated the configuration change are already available for controlling the recovery.
2. The MVS algorithm for determining the incremental set of configuration changes is deterministic, always producing the same results for a particular set of input I/O configuration definitions.

The DRM primitives provided by the hardware allow the software to keep state information in the channel subsystem (**Figure 8**) while the system undergoes the transition from the source configuration definition to the target configuration definition. If a system failure occurs (e.g., a DRM software failure, an MVS error that causes DRM to fail, an MVS error that causes MVS to fail, an LPAR failure that causes the logical partition to fail, or a machine error that requires a re-IPL of the software but not a POR of the machine), the state information kept by DRM provides MVS with enough information to determine which IODF contained the source I/O configuration definition and which contained the target I/O configuration definition. Additionally, MVS can determine how far the configuration change proceeded, and the direction in which the configuration change was progressing (the normal source-to-target transition, or backing out toward the source). In invoking the recovery function of DRM, the installation has the option of overriding the default on the basis of its requirements and changes in environment (e.g., the reason for a back-out has been corrected). The default is to continue in the same direction as when the failure occurred. Recovery of a system failure while making a dynamic configuration change is required prior to any additional DRM changes.

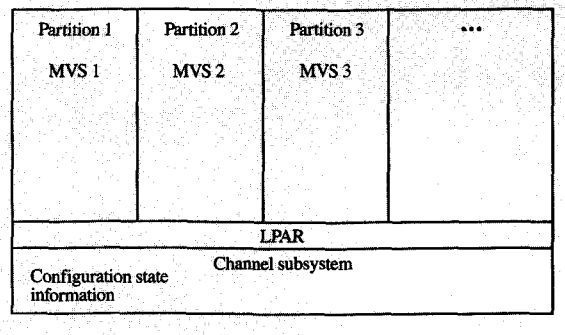

Figure 8

Recovery of dynamic changes. State information kept in the channel subsystem allows MVS DRM to continue dynamic I/O configuration changes that may have been interrupted by a system outage. The changes can be continued after a restart of a partition or from another partition running MVS DRM.

Conclusion

The Dynamic Reconfiguration Management function of MVS/ESA for ES/9000 processors provides an important step in fulfilling continuous availability requirements. The implementation is an evolutionary step toward the automatic management of I/O configurations. Its design provides for simplicity of use, robustness of function, minimization of disruption, and integration of recoverability.

MVS/ESA, Enterprise System/9000, ESCON, Hardware Configuration Definition, ESCON Manager, System/360, and

ES/9000 are trademarks, and System/390 is a registered trademark, of International Business Machines Corporation.

References

1. *MVS/ESA Planning: Dynamic I/O Configuration*, Order No. GC28-1674; available through IBM branch offices.
2. *MVS Configuration Program*, Order No. GC28-1615; available through IBM branch offices.
3. *Hardware Configuration Definition: Using the Dialog*, Order No. GC33-6457; available through IBM branch offices.

Received May 9, 1990; accepted for publication January 15, 1992

Richard Cwiakala *IBM Enterprise Systems, P.O. Box 950, Poughkeepsie, New York 12602 (CWIAKALA at POKVMCR3).*

Mr. Cwiakala is a senior programmer at the Mid-Hudson Valley Programming Laboratory. He received his B.A. in mathematics from Kean College, Union, New Jersey, in 1965 and joined IBM in 1968 at the Poughkeepsie Programming Center in Poughkeepsie, New York. He has held a variety of test, development, design, and management assignments in the MVS system control program. Mr. Cwiakala received an IBM Outstanding Innovation Award for his work on Dynamic Reconfiguration Management and two IBM Invention Achievement Awards; he is a co-inventor on six IBM patent applications.

Jeffrey D. Haggar *IBM Enterprise Systems, P.O. Box 950, Poughkeepsie, New York 12602 (HAGGAR at POKVMCR3).*

Mr. Haggar is an advisory programmer at the Mid-Hudson Valley Programming Laboratory. Since joining IBM as a programmer in 1983, he has worked in the design and development of the MVS operating system. He received a B.S. in computer science in 1983 from Rensselaer Polytechnic Institute, Troy, New York. Mr. Haggar received an IBM Outstanding Technical Achievement Award for his work on Dynamic Reconfiguration Management; he is a co-inventor on three IBM patent applications.

Harry M. Yudenfriend *IBM Enterprise Systems, P.O. Box 950, Poughkeepsie, New York 12602 (HARRYY at POKVMCR3, harryy@pokvmcr3.vnet.ibm.com).*

Mr. Yudenfriend is a senior programmer at the Mid-Hudson Valley Programming Laboratory. He joined IBM in the Poughkeepsie Programming Center as a junior programmer in 1980. He has held various design and development responsibilities in the MVS operating system for the creation of and system support for I/O architecture enhancements on System/370™ and System/390, including ESCON and Dynamic Reconfiguration Management. He has received an IBM Outstanding Innovation Award for his work on Dynamic Reconfiguration Management, and three IBM Invention Achievement Awards. He is also a co-inventor on nine IBM patent applications. Mr. Yudenfriend received a B.S. in computer science from the Columbia University School of Engineering and Applied Science in 1980. He has been a member of the ACM since 1979 and an affiliate member of the IEEE Computer Society since 1981.

System/370 is a trademark of International Business Machines Corporation.