hp availability stats and performance software: full availability monitoring

white paper

When service-level availability is monitored with specific goals in mind, the actual availability of application services is significantly improved. To achieve and maintain continuous service availability levels, both operations management and business analysts must adopt goal-oriented, objectives-based monitoring for computer system components as well as abstract application domains. It is not sufficient to provide high availability hardware and operating system software. Continuous application service levels can only be assured if internal application domains and metrics are also monitored. When availability is inadequately monitored, degraded service levels go undetected, and decreased availability or total unavailability results.

contents

invent

overview

HP Availability Stats and Performance (ASAP) software was developed to provide a uniquely integrated, extensible infrastructure for monitoring the availability and performance of system and application objects.

ASAP software integrates both availability and performance information to form normalized availability vectors for monitored domains and associated properties. Information integration includes operational status, performance, and availability objectives for HP NonStop™ systems, subsystems, and abstract application domains.

functional layers

ASAP implementation consists of three functional layers: Client, Server, and Extension. The features of these architectural layers are described in the text that follows.

ASAP client

The ASAP Client provides visualization and in-depth analysis of object state information. The ASAP Client communicates with the ASAP Server and ASAP Extension (ASAPX) software running on the NonStop Kernel operating system. The ASAP Client provides

- A graphical user interface (GUI) with real-time, state-propagating hierarchical views
- 3-D, color-coded availability-vector graphs with context-sensitive drill-down reporting
- State Change scoreboards and logs with context-sensitive historical drill-down reporting
- Late/early statistics validation for the detection of nonresponding objects and domains
- Full customization of monitored entities and availability vectors defining object availability
- Historical object cache and download wizard for historical analysis of availability data
- Interactive Development Environment for the development of new entity definitions
- Rich graphical interface that executes on Microsoft® Windows NT, Windows 2000, Windows 95, Windows 98, and Windows Me
- Intelligent agent functions that allow object state information to be forwarded and shared with enterprise management frameworks

ASAP server

The ASAP Server executes on the NonStop Kernel operating system. The ASAP Server monitors, gathers, and analyzes availability, state, and performance statistics throughout a network of NonStop clusters. The ASAP Server provides the following features:

- Statistics Gathering Processes (SGPs) that collect availability and performance information
- NonStop ASAP Monitor process pairs that maintain persistent SGPs on each node
  - SGPs are designed to collect statistics with extremely low performance overhead
  - SGPs forward object state information to NonStop ASAP Database Collectors (ASAP Collectors store availability data in a Third-Normal-Form, real-time, historical database.)
- Ability to be configured to operate autonomously on a single node
- Ability to operate collaboratively to form superclusters of availability vector information
- Optional internode time-of-day synchronization between monitored nodes
- Monitoring of availability and performance of systems, subsystems, and applications
- System objects that include CPU, Disk, HP Expand software, Expand/IP, Node, System, and Tape
- Subsystems that include File, Subvolume, Process, HP NonStop Remote Database Facility (NonStop RDF) software, Spooler, and HP NonStop Transaction Management Facility (NonStop TMF) software
- Applications that can be externally monitored using the Process entity
- Application domains that can be internally monitored using the ASAPX API
- Selection of monitored objects that can be done either via autodiscovery or manual object selection
- Both upper- and lower-bound Discrete Object Thresholds (DOTs) monitoring
  - DOTs can be set on classes, subdomains, individual objects, and hierarchical combinations
  - DOTs are permanently remembered in a fully audited objectives database [1]

[1] Note: Objectives database can be NonStop TMF protected, but NonStop TMF software is not mandatory.

  - DOTs can optionally generate traps or Event Management Service (EMS) events
- Interfaces to the ASAP Extension API, allowing customer applications to benefit from the ASAP Client/Server infrastructure when application programs use the optional ASAPX product

ASAP extension software

The ASAP Extension (ASAPX) software executes on the NonStop Kernel operating system.[2] ASAPX software provides an application program interface (API) into the ASAP infrastructure so that the availability and performance of abstract application domains can be monitored. The ASAPX product

- Allows application domain statistics to be integrated with the ASAP Client/Server infrastructure
- Lets application programs benefit from the same ASAP architecture that is used for HP objects
- Collects statistics via an ultra-high-performance, shared-memory architecture
- Provides measurement, viewing, and analysis of application service-level objectives
- Automatically evaluates predefined objectives to establish alert priorities
- Tracks the productivity, performance, and availability of applications
- Tells operators and administrators when application processes do not meet objectives

In addition to the functional layers described here, ASAP software includes additional architectural features that will be described later in this paper.

[2] ASAPX software is an optional product in the ASAP product family.

database

ASAP software includes a database that encapsulates both statistical and service-level objective information. Statistical information includes availability, statistics, and performance data. Objectives information includes user specifications about which objects should be monitored and the service-level objectives for monitored objects.

statistics data

The ASAP statistical database contains both current and historical availability, statistics, and performance information. The database is self-maintaining; it can be configured to retain various amounts of data based on user-defined options. It can retain information indefinitely, purge data daily at midnight, or roll over database files at a preconfigured time of day. Because the database contains both current and historical availability data, the database is used for both online and historical reporting.

The ASAP database schema is published and can therefore be accessed by customers and third parties for real-time or batch-query statistical analysis. Because the ASAP database is normalized in Third Normal Form, and because it can be fully partitioned, the database lends itself to high-performance superscaling for applications in which continuous online monitoring and archival of high-availability information is a requirement.

objectives data

In addition to statistical data, the ASAP database also contains user-configured, object-selection, and availability-objectives criteria. Objectives are stored in a fully audited, recoverable database. Both the selection of objects and their service-level objective settings are permanently retained after CPU reloads, or in the extraordinary event of a catastrophic system failure. Service-level objectives contained in the database support both upper- and lower-bound specifications; for example, Busy > 10 and Busy < 60, as well as multiple property objectives, such as RANK Process $XYZ, Busy > 10, Busy < 60, QLen < 7, and Pri = 150.

Database objectives permit both individual domain objectives as well as hierarchical specification. For example, you can specify that all disks should be less than 60 percent busy, but that certain disks should be treated differently, for example, and be less than 30 percent busy.

entity definition language

ASAP Client, Server, and Extension components incorporate an Entity Definition Language (EDL) that provides extensible definition of abstract entities and attributes as they relate to ASAP software features and functions. EDL allows system entities, customer application domains, and third-party entities to be defined externally to the ASAP software environment.

entities, attributes, and data

The notions of entity and attribute in ASAP software are somewhat synonymous with the notion of table and column in the SQL data model. An *entity* can be thought of as a table, while an *attribute* can be thought of as a column in a given table. ASAP software differs from the SQL model in that entities and attributes have intrinsic properties that relate specifically to ASAP software's features and functions. EDL also allows data to be included in an EDL file, so an EDL file can represent the encapsulation of entity-attribute schema, statistics, and state information for running systems.

interactive development environment

The ASAP Client includes an Interactive Development Environment (IDE) that is used by software developers to interactively develop system and application entity definitions. The IDE includes context-sensitive interactive help for the EDL. IDE includes functions that allow EDL environments to be edited, compiled, exported, and imported. The EDL IDE also allows data from live host sessions, along with entity definitions, to be interactively saved.

discrete object thresholds

All entities and attributes monitored by ASAP software are defined using the EDL. The entire ASAP service-level objectives namespace is defined by EDL. Each entity that is defined to ASAP software using EDL is an entity that can be monitored by ASAP software.

ASAPX software

EDL also defines actions performed by the ASAPX software on behalf of application entities. This includes how ASAPX software defines and treats application data in shared memory, how it computes the values of individual attributes, as well as the format of the record produced for an application in the ASAP database.

EDL environments

On the Microsoft Windows system, EDL files are text files with an "EDL" file extension. An EDL file contains one or more EDL statements, and thus the file provides a portable container representing an ASAP environment.

sharing environments

Users and developers can share EDL "environments" with other users and workstations. Copying or mailing an EDL file to another workstation provides the sharing of schema, statistics, and state information. Because EDL is a registered extension in the Microsoft Windows system, double-clicking an EDL file within Outlook, for example, redisplays the original environment that created the EDL.

capturing environments

EDL files can contain data representing availability, statistics, and state information for a running environment. For example, when an ASAP Client is monitoring system and application objects, the schema, statistics, and state information for the entire environment can be saved to an EDL file simply by clicking the "save" toolbar button.

Capturing entire system and application environments using EDL is beneficial because it

- Provides portable encapsulation of application and system entity definitions
- Allows entities, attributes, states, and statistics to be stored in a single EDL file
- Allows different sets of customized ASAP settings
- Allows environments (and data) to be shared

EDL benefits

EDL allows environments that represent observed system and application behavior to be captured and shared. This capability has been shown to be useful for

- *Problem reporting*—captures environments, including statistics and state information
- *Prototyping*—system or application entity, attribute, and data
- *Collaboration*—group discussion of observed behaviors
- *Education*—demonstration features of an environment
- *QA and testing*—reproducing test scenarios

monitoring

Consistent with continuous availability requirements, ASAP software allows dynamic selection of monitored objects. Monitored objects can be added or removed while the ASAP system is running. If no objects are selected for an entity class, the SGP for that entity will automatically configure a set of objects. For example, if you do not specify a CPU to be monitored, then all CPUs will be monitored.

selection

The ASAP MONITOR command[3] allows you to specify which objects should be monitored. Specific ASAP entities, such as CPU, Disk, Expand, File, Process, Process Busy, NonStop RDF, Spooler, Tape, and TMF, can be selectively monitored using the MONITOR command. For example, if you want to externally monitor Processes $App, $Funds, and $Atms, you would use the following MONITOR commands:

+ MONITOR PROCESS $App

+ MONITOR PROCESS $Funds

+ MONITOR PROCESS $Atms

+ COMMIT

Also consistent with continuous availability requirements, objects can be removed from service at any time. For example, monitoring of process $App could either be temporarily suspended or stopped altogether with either of the following commands. To suspend monitoring and keep objects in the database, the following command is used:

+ MONITOR PROCESS $App, OFF

To remove objects from the objectives database, the following command is used:

+ MONITOR PROCESS $App, DELETE

commit processing

When you change objects and associated objectives, they are permanently saved in the ASAP objectives database. Database changes are not applied to ASAP monitoring components until you "commit" changes using the COMMIT command.[4] The COMMIT command tells ASAP SGPs to reload their objectives from the ASAP objectives database. The COMMIT processing architecture used by ASAP software allows batched, instantaneous alteration of large numbers of objects in conjunction with efficient dynamic reconfiguration of objectives. This allows the entire monitoring configuration to be instantaneous and radically changed at various times of the day.

objectives

ASAP software provides a rich set of service-level objective capabilities. The software's service-level objective features are often referred to as Discrete Object Thresholds, or simply DOTs. The DOTs acronym is useful because it helps users visualize the software's ability to focus on a specific dot in the huge universe of objects and properties.

DOTs provide both upper- and lower-bound objectives, for example, Busy > 30 and Busy < 50, as well as complex specification of availability property objectives for multiple attributes. DOTs can be set on classes, subdomains, and individual objects and are permanently retained in the audited objectives database.

ranking objects

The ASAP RANK command[5] allows you to specify DOTs for monitored objects. Any data attribute defined in ASAP software that has a state pair vector can be "ranked." For example, if you want the performance objective for CPUs to be less than 50 percent busy, and to have a queue of less than 6 processes waiting to execute, you would use the following command:

+ RANK CPU, BUSY < 50, QUEUE < 6[6]

[3] See section 5 of the ASAP Server manual.

[4] For more information about the ASAP COMMIT command, see section 5 of the ASAP Server manual.

[5] For more information about the ASAP RANK command, see section 5 of the ASAP Server manual.

[6] The ASAP COMMIT command applies Monitor/Rank objective changes.

service-level objectives

The ASAP RANK command makes a statement about the preferred service-level objectives for an object or a class of objects. For example, the statement

RANK ATM Chicago, CASH > 2000

asserts the objective that ATM machines should contain more than US$2000. Service-level objectives can also be defined for hierarchical domains. For example, if you would like a different objective for ATMs in the Chicago Loop, you could use the following syntax.

For the objective for all ATMS, the following command could be used:

+ RANK ATM, CASH > 1000

For the objective for all Chicago ATMs, the following command could be used:

+ RANK ATM Chicago, CASH > 2000

For the objective for Chicago Loop ATMs, the following command could be used:

+ RANK ATM Chicago\Loop, CASH > 4000

When service-level objective thresholds are not met, alerts are displayed in the following ways:

- Graphically in the ASAP Client with color-coded icons, graphs, logs, tree, and list views
- Optionally through tokenized EMS subsystem events in standard NonStop Kernel operating system EMS logs
- Graphically in the Open Enterprise Management (OEM) gateway
- Graphically in participating Enterprise Management Frameworks (EMF)

objective commands

ASAP software provides two objective commands, MONITOR and RANK, which are used to specify object selection and service-level objectives for system objects, subsystems, and application domains. Following are examples of possible objectives.

To monitor a DISK named $DATA, you would use the following syntax:

+ MONITOR DISK $DATA

To set objectives for DISK $DATA to be less than 80 percent full, you would use the following syntax:

+ RANK DISK $DATA, FULL < 80

To set objectives for DISK volume $DATA to be less than 80 percent full, to be less than 30 percent busy, and to have a request queue length (QLEN) less than 5, you would use the following RANK syntax:

+ RANK DISK $DATA, FULL < 80, BUSY < 30, QLEN < 5

To set objectives for DISK volume $DATA to have its primary CPU be equal to 1, the Primary disk (P) controller path be the Primary “P” path, and for the Mirror disk (M) controller path be the Backup “B” path, the syntax would be

+ RANK DISK $DATA, CPU = 1, P = “P”, M = “B”

To monitor the file $System.System.Userid, set the service-level availability objectives to be owned by “255,255”, and to be secured with an RWEP security vector of “OOOO”, the syntax would be

+ MONITOR FILE $System.System.Userid

+ RANK FILE $System.System.Userid, OWNER="255,255", RWEP=“OOOO”

To set the service-level availability objectives of all CPUs to be less than 70 percent busy, except for CPU 5, which should make its objectives be less than 55 percent busy, and its queue less than 4, the syntax would be

+ RANK CPU, Busy < 70

+ RANK CPU 5, Busy < 55, Queue < 4

To set the service-level availability objectives for an ATM machine named “NorthMain” to have more than US$2000, to specify its transaction rate should be less than 3 transactions per interval, and to specify that bad PIN verifications should be less than 4 per sample, the syntax would be

+ RANK ATM NorthMain, CASH > 2000, Trans < 3, BadPins < 4

To monitor a running process $FUNDS, assure its continuous availability, set its service-level availability objectives to be less than 25 percent busy, its request queue length (QLEN) less than 5 requests, its primary CPU = 1, and its Priority > 150, the syntax would be

+ MONITOR PROCESS $FUNDS

+ RANK PROCESS $FUNDS, Busy < 25, QLEN < 5, Cpu = 1, Pri > 150

To monitor the NonStop Remote Database Facility (RDF) subsystem between NonStop clusters \Chicago and \Newyork, and set the service-level availability objective for the audit trail Relative-Time-Delay to be less than 90 seconds, the syntax would be

+ MONITOR RDF Chicago->Newyork

+ RANK RDF, RtdSecs < 90

objective events

ASAP software provides EMS events or "trap" generation when service-level objectives are not met. Very fine control of event generation is provided. You can specify event generation down to a specific objective for a specific object. For example, the following syntax would "repeatedly" generate "critical" EMS events if the RWEP is not "OOOO" for the system Userid file:

+ RANK FILE $System.System.Userid, RWEP = "OOOO" CRITICAL REPEAT

The following would generate an EMS event when the priority of process $XYZ was not 150, but only one event would be generated, it would be an informative event (INFO), and it would not be displayed on the operator console (for example, only for programmed automation purposes):

+ RANK PROCESS $XYZ, Pri = 150 INFO NODISPLAY

ASAP extension software

Externally monitored applications cannot meet extreme continuous availability requirements. Even well-coded applications that generate alerts at all the proper times can get hung, go into processing loops where alerts aren't generated, or be affected by hardware and other external problems. And while such applications may appear fine externally, internally they are not, and the services they provide are degraded.

ASAPX Extension (ASAPX) software monitors applications by looking directly inside application programs. By inserting its API procedures into main processing loops, ASAPX software knows immediately when an error branch has been taken in the code, or when a successful transaction is completed. Application service levels can be directly measured in real time. Using DOTs, ASAP users can be alerted when service levels begin to degrade before problems become serious.

using shared memory

ASAPX software has no measurable impact on application performance. It accomplishes its monitoring function by using shared memory in each processor as the primary communication medium between itself and the application. A monitor process in each processor allocates an extensible flat memory segment that is shared by the ASAPX process and by all applications using its API. Application updates to counters and other values are direct memory updates. Other than the initial registration message to the monitor process, there is no interprocess communication.

ASAPX software tightly controls shared memory. The ASAPX monitor process and API create and maintain boundary tags and checksums to ensure consistency of all application data allocated in shared memory. Boundary tags are used to uniquely identify an application domain and to protect against cross-boundary move operations. Checksums are computed before and after each memory update to ensure data consistency.

producing application statistics

ASAPX software samples application data at each interval by reading its shared-memory segment and performing user-defined computations to produce the values reported for each domain. Data records are generated and written to ASAP Collector processes for storing into the ASAP database.

ASAPX domains are abstract representations of application services. They are long free-form names with up to five levels of hierarchy. For example, an ATM in Chicago might be represented as "Atm\Chicago\$Atm1", "Atm\Chicago\Deposit\$Atm1", or Deposit\Chicago\$Atm1".

The first (leftmost) level of an ASAPX domain name is the name of an entity, as defined in EDL. EDL entities can represent user applications, or a subset or superset of applications, depending on the view the user establishes.

custom attributes

The actions taken for an entity by ASAPX software, from the way the software defines and treats application data in shared memory to the format of the record produced for the application, is defined by EDL. At the entity level, users define the content of shared memory and other entity-wide properties. At the attribute level, users define the attributes to be produced, including the formulas used to produce the attribute values at each interval.

Using EDL, ASAPX software performs custom calculations for each attribute and entity, producing entity-specific records that are sent to the ASAP Collectors for storing in the ASAP database. For example, an entity might define two items in shared memory, a transaction count, and an error count. The ASAPX API could be used within the application to update the transaction count after each successful transaction (ENDTRANSACTION), and the error count after each failed transaction (ABORTTRANSACTION).

The two counters could be used to produce a variety of attributes to determine the application service levels. Some of the attributes that could be defined are a transaction count, a transaction rate, an error count, an error rate, a success percentage, a fail percentage, a total transaction count, and a total transaction rate.

ASAPX software provides 11 built-in attributes that can be included in an application EDL file. These attributes are automatically computed and formatted by the software when it finds them in application EDL. ASAPX software's built-in attributes are shown in the following table.

ASAPX software's built-in attributes

attribute name	description
Avail	Availability of the application domain since it first registered with ASAPX software
Busy	Percent CPU busy for the application process that registered the domain
Cpu	Processor where the registering process executes
DownTime	Total downtime in seconds since the domain was first registered
Nak	Number of intervals since the application domain was last updated
Pri	Execution priority of the registering process
Pstate	Process state of the registering process
RegTime	Initial date/time domain registered with ASAPX software
UnAvail	Unavailability of the domain for the interval
Version	Version used by the application when registering with ASAPX software
WState	Wait state of the registering process

data aggregation

ASAPX software can aggregate data across any level of an application domain name. For example, if application domains used the form "Atm\Chicago\Deposit\$Atm1", then ASAPX software can produce an aggregate total record for each of the three upper levels of the name "Atm", "Atm\Chicago", and "Atm\Chicago\Deposit".

The software automatically propagates the worst state from the aggregate set for each attribute to the aggregate record unless a specific objective has been defined for the attribute at the aggregate level. This enables users to monitor the combined service levels of all application components in a single application view.

simplicity and ease of use

The ASAPX API is simple and easy to understand. There are a total of six API procedures, but only two are required for any application. First, the application calls the ASAP_REGISTER_ procedure to register with ASAPX software, then it calls one of the update procedures, ASAP_UPDATE_ or ASAP_UPDATELIST_, to update one or many of its items in memory at the appropriate points in the application code. The application might also call the ASAP_REMOVE_ procedure to remove itself if it is operating in batch mode, or it might remove and reregister if ASAPX software returns an error from one of the API calls.

availability

It's important to understand that the definition of service-level availability is dependent on a user's point of view. As a result, ASAP software must provide customization of availability and state propagation algorithms.

point of view

One view of availability is the centrally administered view. In this view, there is one central "policy" that governs the set of availability objective rules. This type of policy is seen in "glass room" environments, where operations management personnel dictate the definition of availability.

However, availability can also be a function of each individual user's point of view. For example, if you are at the front of a line, your view of service availability is quite different than that of a person who is at the back of a 50-person line. There are other important ways that the definition of availability depends on user point of view. For example, a financial analyst monitoring the availability of a funds transfer application has a totally different notion of availability criteria than an operator who is worried about the state of physical objects, such as controller paths and mirrored disk drives.

In a large enterprise, the number of business analysts who require "point-of-view" availability monitoring can easily outnumber the people in the operations management staff. Thus, "point-of-view" availability represents an important class of availability monitoring. There may be dozens of analysts at workstations throughout a large enterprise who are monitoring "availability." Each of these users can have an entirely different view of availability based on his or her own definition of availability. Thus, different users must be able to customize their definition of object availability.

If GUI availability displays are to be useful, then it must be possible to segregate the information in them. For example, a financial analyst may need to know whether the total dollar amount transferred in the last hour in a FUNDS transfer application met expectations, while another person may need to know how many ATM transactions have occurred. Most often, neither care about each other's notion of availability, and they don't typically want their color-coded alert displays to be cluttered with alert information based on another person's notion of availability.

Customizing availability must also allow users to include and exclude properties that may be used to determine availability. For example, many users may care whether a certain application process is running, but only a few may care whether the application runs in CPU 15.

In order to address such requirements, ASAP software provides a range of customization options that define which entities and attributes are analyzed, as well as the state determination rules used to determine availability. These properties and rules can be customized in the software, and thus can be used to control the state propagation engine.

If the general notion of availability were a fixed concept, state determination and propagation would be a relatively straightforward process. However, different users can and do have different definitions of service-level availability. In fact, a user's definition of availability can change many times during the day. The ASAP Client, Server, and Extension architecture is designed to operate with these requirements in mind.

state determination

In the previous section, we discussed why there cannot be a single definition of availability for all users. Furthermore, state determination rules can and do change for a single user in a matter of seconds. One instant a user may view availability in terms of whether the execution priority and preferred CPU number for application processes meet their objectives. The next minute that same user may not want these attributes factored into state determination at all, but rather may need to view state determination for processes based on busy utilization and queue lengths.

Because the definition of availability is not fixed, it follows that object state determination rules must be variable. Thus, ASAP software must be able to instantly change its state determination rules based on a user's relative notion of availability at any given moment. For example, one instant it may mean using one set of attributes, and at another instant it may mean including or excluding a different set of object attributes. These capabilities must be provided for both centrally administered policies, as well as for customized user availability definitions.

As a result of these requirements, ASAP software provides customized state determination based on

1) Central policy database objectives, as described in the earlier "Service-level objectives" section

2) Customized user-defined state determination rules

3) Combinations of both 1 and 2

Using these customization techniques, object classes such as the process entity can have the state determination for its priority attribute based on centrally administered policy, but have the state of its busy attribute based on an individual user's customized thresholds. The computation of the overall state of the object is based on combinations of selected attributes, and by analyzing unique state determination rules assigned to each entity attribute.

ASAP software provides the following types of object-attribute state determination rules:

- Allow inclusion and exclusion of attributes for the determination of an object's state
- Use centrally administered objectives for an attribute's state determination
- Use custom user-specified thresholds for an attribute's state determination
- Use application-assigned state for an attribute's state determination

Because the definition of availability depends on a user's point of view, ASAP software is designed to allow users to define the rules that determine the state of each object attribute, integrate attribute states to determine the overall state of each object, and propagate object state information upward through object class hierarchies.

state propagation

ASAP software propagates object states upward through object class hierarchies. To do this, ASAP software analyzes each object's attributes, their state determination rules, their metric values, and compares those values with upper- and lower-bound service-level objectives. As each attribute is analyzed, it is assigned an availability vector, or state. Examples of such vectors or states might include "OK" or "Warning." When all the attributes for an object have been analyzed, ASAP software can make an overall statement about the state of the object. Once the state of an object is determined, object states are propagated upward through the object class hierarchy for that object.

For example, if an ATM in the Chicago Loop named ATM\Chicago\Loop has insufficient funds, and the lower-bound objective for that property is not met, the state of that object can be said to be in a "Warning" state. As the state is propagated to higher levels of the object hierarchy, the status of all ATMs in Chicago inherits the Warning state; for example, ATM\Chicago*. Finally, as the state is propagated to the all-ATMs level, ATMs in general can be said to be in a Warning state; for example, ATM*. This overall process is called *state propagation*, and is displayed in a variety of ways in the ASAP Object Integration Layer (OIL) and OEM gateway.

open enterprise management

The HP OEM provides an infrastructure for integrating client monitoring applications, such as ASAP software, with Windows system-based EMFs.

OEM components

The OEM environment is made up of four distinct components, all of which execute on Windows NT, Windows 2000, Windows 95, Windows 98, and Windows Me systems:

- One or more client applications/agents, such as the ASAP Client
- The OEM Server
- One or more EMF adapters
- One or more EMFs

client applications/agents

Client applications/agents serve as data providers to the OEM and integrate directly with OEM Server using the published OEM client APIs. They furnish information on classes, objects, and their associated states to the OEM Server.

The OEM is a very general framework that has no specific knowledge of NonStop Kernel operating system objects, hierarchy, or naming. Clients can therefore truly report state information on any application-defined object and are not limited by any OEM-imposed view of the object space.

Clients may also optionally define and associate one or more actions with each supplied class or object. These actions are passed through to EMF adapters and the EMFs themselves, thereby making it possible for users to invoke the actions directly from the OEM or EMF.

Finally, clients also respond to notifications from the OEM Server whenever defined actions are invoked against specific objects. The clients can in turn perform any processing necessary, based on the action chosen.

OEM server

The OEM Server acts as a central repository and clearinghouse of object and state information. It also serves as an intelligent "traffic cop," routing data and requests between client applications and EMF adapters. The OEM Server provides the following capabilities:

- Maintains an internal database of all defined classes and objects, their states, any details reported by client applications, and all actions defined for those classes and objects.
- Encapsulates the interface to EMFs and isolates client applications from EMFs and EMF behavior. Clients only need to interface with the OEM Server to integrate with all supported EMFs. As the OEM supports new EMFs, all existing client applications are automatically integrated with that EMF without any code changes to the client.
- Encapsulates the interface to clients and hides implementation details from EMF adapters. Each EMF adapter interfaces to the OEM Server and, as a result, is not required to have specific knowledge of how to interface with any particular client application. This allows users of a particular EMF to invoke client-defined actions against objects without requiring custom logic within the adapter to deal with each particular client. Therefore, new clients can be developed without having to change existing EMF adapters.
- Allows integration of EMF state models and client models, such as the following ASAP states: Exists (1), Up (2), Low (3), Medium (4), High (5), Warning (6), Critical (7), Down (8), and Unknown (9).
- Tracks reported state information separately for each client application. This feature, known as *state arbitration*, makes it possible for multiple clients to report state information on the same object and ensures that only the most critical information is forwarded to EMFs. For example, if client application A reports that disk $SYSTEM is in state 7, while client application B reports that $SYSTEM is in state 5, the OEM Server will ensure that the more critical state (7) is passed to the EMFs. If application A later reports that the problem with $SYSTEM has been fixed and the state is now 1, the more critical state (5) reported by client B will be forwarded to the EMFs. This differs from the behavior of many EMFs, which do not distinguish between different client applications and always assume the last reported state is the true state of the object.
- Continuously propagates state information throughout its object hierarchy, making it possible to determine quickly the state of higher-level objects by rolling up the states of subordinate objects. Thus, users can quickly be directed to domains that have problems and can drill down to the specific objects in error.
- Provides standard visualization of all object and state data via Alert windows. Alert windows contain sorted lists of object names and states that range from the most to the least critical. Objects in the most critical states will always “bubble” to the top, making it easy for users to determine which objects are most in need of attention.

- Supports custom views of data, which allow users to group heterogeneous objects based on any criteria they deem appropriate. State values are propagated within the hierarchy of each custom view independently, so that users can get a high-level picture of the state of business objects and subsystems. For example, a customer might group objects by application. Therefore, a bank could have a custom view for their ATM application, another for their Society for Worldwide Interbank Financial Telecommunications (SWIFT) application, and so on. The OEM tracks and reports the status of each of these views independently, allowing customers to track object state by business function rather than from a system standpoint.

enterprise management framework adapters

An EMF adapter is responsible for providing the integration layer between the OEM Server and a single EMF. EMF adapters interface with the OEM Server via the published OEM adapter APIs and interface with the EMF using whatever scheme is necessary for that EMF. The OEM currently supports four adapters: the HP OIL adapter, which provides a hierarchical tree or Windows Explorer view of objects and states; the Unicenter TNG (The Next Generation) adapter, which integrates with the Computer Associates TNG framework; and the IBM Tivoli NetView adapter, which integrates with the Tivoli NetView framework. In addition, customers have developed their own custom adapters using the published OEM adapter API.

An EMF adapter encapsulates the interface to a particular EMF and hides all details of that EMF from the OEM Server and client applications. The adapter supplies the EMF with all OEM object, state, and action data by translating standardized OEM requests into EMF-specific operations and functions.

In addition, the adapter allows users to invoke client-defined actions in the EMF and notifies the OEM Server of the invocation by translating EMF-specific operations to standard OEM requests. This in turn allows the OEM Server to notify the necessary client applications that the user has chosen a specific action and enables the client to respond accordingly.

enterprise management frameworks

EMFs play a role within the OEM environment in that they are the ultimate destination for all client-supplied object and state data. Therefore, they may be true enterprise management frameworks such as Tivoli NetView or CA-Unicenter. However, they could just as easily be a user-supplied management application that performs a very specific function. Because EMFs are entirely hidden from the client applications and OEM Server, there really are no requirements for what does or does not constitute an EMF. The only true requirement is that an EMF adapter be supplied that acts as the bridge between the OEM Server and the EMF. Once this "bridge" is in place, the actual EMF can be as complex or simple as necessary, based on the needs of the user.

The OEM Server also provides its own visualization of object and state data via Alert windows and custom views. Thus, it can also be used without any EMF. The OEM Server, in conjunction with the participating client applications, is capable of supplying both high-level and detailed information about objects and states and presents that information to users in a comprehensible and intuitive fashion.

For more information, go to www.hp.com/go/nonstop.

October 2002, first published July 2001.

02-0412