Mode Management Service
Alex Kirn
akirn@eos.hitc.com

16 April 1996
Mode Management Service Outline

- Driving Requirements
- Software Design
- Physical Design
- User Interfaces
- Data Model
- Object Model
- Mode Management Service Initiation Event Trace
- Reference data
  - HP OpenView and Agent Functionality
  - Main Loop and Callback Routines
Driving Requirements

• ECS shall support simultaneous production and testing activities
• ECS shall provide the capability to initiate a new mode of execution
• ECS shall be capable of monitoring and controlling each mode of operation
# MMS Software Architecture Overview

## MLCI
- SW Distribution Management
- License Management
- Change Request Management
- Inventory/Logistics/Maintenance Management
- SW Change Management
- Baseline Management
- Training Management
- Policies & Procedures Management

## MACI
- Management Agent Services

## MCI
- Mode Management
- Fault Management
- Performance Management
- Report Generation Management
- Security Management
- Accountability Management
- Physical Configuration Management
- Trouble Ticketing
- Management Data Access
- Billing / Accounting
- Management DBMS
- Management Framework (HPOV)
- Enterprise Framework (Tivoli)
- Backup and Restore
- User Comment Survey
MMS Software Architecture Overview (cont.)

Active Modes List

HPOV COTS

Mode Management Service

p/o MCI

Master Agent

Deputy Agent

SubAgent

p/o MACI

Active Modes

(NFS/DFS)

events

gets

traps, sets
MMS Software Architecture

HP OpenView Windows (ovw) COTS

ovwInit()

ovwAPI’s

Mode Management Service (MMS) Application main()

snmpAPI’s

snmp over UDP

MsAgDeputy

snmp via the DOF

Traps and Sets

Managed host

Active Modes

Active Modes List

Managed host
MMS Physical Design

Manager Host

- HPOV COTS
- Mode Management Service
- Deputy Agent
- Active Modes

Managed Host

- Master Agent
- SubAgent
- Managed Application
- DOF

Active Modes

ovw API's

snmp API's

snmp

Active Modes List

(NFS/DFS)
MMS Multi-Session View

Management Console

OVwSessionId = ops:0

Management Station

Management Console

OVwSessionId = ts1:0

map database

ovwdb daemon

Mode Management Service

object database

active modes list

NFS

socket

socket

socket

socket
User Interface

• CDS Browser: manual configuration of CDS namespace for adding/removing mode hierarchies
• HP OpenView GUI:
  – MMS seamlessly incorporated into HPOV’s standard user interface
  – Mode Activation/Deactivation
  – Monitoring: status propagation, event notification
  – Control: Life-cycle; startup, shutdown, suspend, resume
Data Model

• Active Modes List
  – Maintained on the manager host
  – Maintained by Mode Management Service
  – Accessed by Mode Management Service, SubAgent, and MDA
  – one per DCE cell
  – Format:
    » modelIdentifier1 <simTime> <deltaTime>
    » For Example:
      • ops
      • ts1 98:256:13:40:00 +1120:08:22:15
      • ts3 96:298:04:45:00
      • tr1
    » simTime format: yy:ddd:hh:mm:ss
    » deltaTime format: +/-ddddd:hh:mm:ss
      • + future, - past
Object Model

- The following object model will be reviewed

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Diagram Name</th>
<th>Document Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMS</td>
<td>Mode_Mgmt_Object_Model</td>
<td>305 Vol29, Section 6.1.3</td>
</tr>
</tbody>
</table>
Dynamic Model

- The following event trace will be reviewed

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Diagram Name</th>
<th>Document Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode Activation</td>
<td>Mode_Activation_Event_Trace</td>
<td>Contained Within</td>
</tr>
</tbody>
</table>
Mode Activation Event Trace

- Select Activate Mode
- Enter New Mode
- Prompt for new mode
- Validate Mode
- Prompt for simTime
- Set simTime
- Add mode to current mode list
- Call API(Activate Mode mode)
- Activate Mode (mode) RPC
- Discover Now

Refer to the Agent Startup Event Trace for more detail on icon registration and normal executable startup.

Register mode specific executable with HPOV.
Event Trace Description

• Assumptions
  • The Mode Management Service (MMS) has already been initiated.
  • The mode entered is “ts1”. (The application will prompt the user for a simulation time after the mode identifier has been entered and validated for all non-ops modes.)

• Event Flow
  • The operator selects an action from within the HPOV GUI to Activate a new mode.
  • Upon instantiation the MsMmModelInit object prompts for the mode identifier and then validates the entry.
  • The valid mode is then set in the MSMMmMode base class and added to the current active modes file.
  • An activate mode ovsnmp API call is issued to the Deputy Agent.
The Deputy Agent uses the DOF to send the call to the MsAgDiscoverer.

MsAgDiscoverer issues a DiscoverNow() call to load the mode specific configuration files based on the active modes as listed in the active modes file.

The information in these files is loaded into the MsAgTeVerExec tables (along with the mode, and simulation time if provided). The executable object(s) are then registered within HPOV and displayed on the appropriate submaps as symbols. (This step and subsequent steps are detailed in the Agent Startup Event Flow).
ovw Background

• Terminology
  – Object - an ovw object represents a real world entity, e.g. a host, an application, a program, a process, etc.
  – Symbol - an ovw symbol is the graphical representation of an object that can be displayed on a map or submap.

• Functionality
  – Object information (which is configurable) is stored locally in the ovw Object Database.
  – An object can be represented by multiple symbols, however a single symbol can represent only one object.
  – ovw triggers events and executes callbacks based on predefined actions (similar to X-windows main loop processing).
  – Each mode will be represented in a separate HPOV session.
  – Each HPOV Session will contain it’s own copy of the ovw GUI. They all share the same ovw object Database and ovw map database. The ovwGUI API calls will determine which session to display a symbol on based the associated object’s mode.
  – Note: trapd filtering capabilities will not be available until HPOV4.1. Our ability to filter realtime events based on mode will be limited until that time.
Design Notes

• HPOV <-> Agent correlation
  – Agents/SubAgents only recognize Application, Program, Process level management commands.
  – HPOV will have ovw symbols that represent system and subsystem level entities.
  – HPOV custom code applications must convert requests to agent level management commands.
  – When the SubAgent finds an application that is new it will send an event to HPOV to add an object to the ovw Object database.
  – The ovw object will be populated with all of the appl, prog, or exec associated information. (e.g. tableIndex, tableID, execName, execID, parentID, hostname, mode, etc.)
  – A corresponding symbol will be added to the applicable submap.
MMS Main Loop

Basic Main Loop Structure:

```c
main(argc, argv)
{
    ....
    ovw initializations
    ....
    ovsnmp initializations
    ....
    add callbacks
    ....
    mainEventLoop(....)
}
```

Basic MMS Callbacks:

```c
ovwActivateMode()
ovwDeactivateMode()
ovwShutdownExec()
ovwSuspendExec()
ovwResumeExec()
```

main() is executed is entered when “Start MMS” is selected from the HPOV GUI. Until that time all other MMS options will be greyed out from them HPOV menus. After the mainEventLoop() has been entered, the MMS options will be activated. When an option is selected the callback will be invoked.
Callback Overview

OVwActivateMode()
{
    // instantiate a mode init object
    MsMmModeInit newMode;

    // call member function to tell agent to activate this new mode
    newMode.ActivateMode();
}

OVwDeactivateMode()
{
    // instantiate a mode term object with the mode to deactivate
    MsMmModeTerm killMode(RWCString mode);

    // call member function to see if valid and tell agent to deactivate mode
    killMode.DeactivateMode();
}
Callback Overview (cont.)

OVwSuspendExec(ovwObject *objectptr)
{
   // create an application level object pointer if needed
   ovwObject *appObject;
   EcTint seconds;

   // obtain number of seconds until suspend
   ovwAPIGUI("Enter Number of seconds until suspend: ", &seconds);

   // system and subsystem level ovwObjects need to be broken
   // down into application level objects.
   if (objectptr->objLevel == "system" || objectptr->objLevel == "subsystem")
   {
      traverse down system or subsystem object tree, then
      for each (appObject = objectptr->application level object) do
      {
         // create a suspend object for each application level ovw object
         MsMmSuspend   suspendObj(appObject->nTblID, appObject->rowIndex,
                                 appObject->hostID, appObject->seconds);

         // issue suspend to agent
         suspendObj.supendExec();
      }
   }
   else
   {
      // create a suspend object for the (app, prog, or process level) ovw object
      MsMmSuspend   suspendObj(objectptr->nTblID, objectptr->rowIndex,
                              objectptr->hostID, objectptr->seconds);

      // issue suspend to agent
      suspendObj.supendExec();
   }
}
Callback Overview (cont.)

OVwStartupExec(ovwObject *objectptr)
{
    // create an application level object pointer if needed
    ovwObject *appObject;
    EcTint seconds;
    RWTime deltaTime = 0;
    RWTime simTime;

    // look for a simTime or deltaTime in the active modes file
    GetSimDelta(objectptr->mode, &deltaTime, &simTime);

    // if no delta entry and there is a simTime entry then calculate delta time
    if (!deltaTime && simTime) {
        deltaTime = CalculateDelta(simTime);
        SaveDelta(deltaTime);
    }

    // system and subsystem level ovwObjects need to be broken
    // down into application level objects.
    if (objectptr->objLevel == "system" || objectptr->objLevel == "subsystem") {
        traverse down system or subsystem object tree, then
        for each (appObject = objectptr->application level object) do {
            // create a startup object for each application level ovw object
            MsMmStartup startupObj(appObject->nTblID, appObject->rowIndex,
                                    appObject->hostID, appMode->mode,
                                    deltaTime);

            // issue startup to agent
            startupObj.startupExec();
        }
    }
    else {
        // create a startup object for the (app, prog, or process level) ovw object
        MsMmStartup startupObj(objectptr->nTblID, objectptr->rowIndex,
                               objectptr->hostID, appMode->mode,
                               deltaTime);

        // issue startup to agent
        startupObj.startupExec();
    }
}
Callback Overview (cont.)

OVwShutdownExec(ovwObject *objectptr)
{
    // create an application level object pointer if needed
    ovwObject *appObject;
    EcTint seconds;

    // obtain number of seconds until shutdown
    ovwAPIGUI("Enter Number of seconds until shutdown: ", &seconds);

    // system and subsystem level ovwObjects need to be broken
    // down into application level objects.
    if (objectptr->objLevel == "system" || objectptr->objLevel == "subsystem")
    {
        traverse down system or subsystem object tree, then
        for each (appObject = objectptr->application level object) do
        {
            // create a shutdown object for each application level ovw object
            MsMmShutdown shutdownObj(appObject->nTblID, appObject->rowIndex,
                                      appObject->hostID, appObject->seconds);

            // issue shutdown to agent
            shutdownObj.shutdownExec();
        }
    }
    else
    {
        // create a shutdown object for the (app, prog, or process level) ovw object
        MsMmShutdown shutdownObj(objectptr->nTblID, objectptr->rowIndex,
                                  objectptr->hostID, objectptr->seconds);

        // issue shutdown to agent
        shutdownObj.shutdownExec();
    }
}
Callback Overview (cont.)

OVwResumeExec(ovwObject *objectptr)
{
    // create an application level object pointer if needed
    ovwObject *appObject;

    // system and subsystem level ovwObjects need to be broken
    // down into application level objects.
    if (objectptr->objLevel == "system" || objectptr->objLevel == "subsystem")
    {
        traverse down system or subsystem object tree, then
        for each (appObject = objectptr->application level object) do
        {
            // create a resume object for each application level ovw object
            MsMmResume    resumeObj(appObject->nTblID, appObject->rowIndex,
                                     appObject->hostID);
            // issue resume to agent
            resumeObj.resumeExec();
        }
    }
    else
    {
        // create a resume object for the (app, prog, or process level) ovw object
        MsMmResume    resumeObj(objectptr->nTblID, objectptr->rowIndex,
                                 objectptr->hostID);
        // issue resume to agent
        resumeObj.resumeExec();
    }
}
Work Flow Analysis

• Testing of Data Server Modification within operational environment
  – Allocate Mode Identifier
  – Develop Plan / Identify Resources
  – Configure System
  – Activate System for new mode
  – Initiate Test
  – Return System to Desired State
Mode Management Support
Points of View I

DAAC Resource Manager

1. Informs SMC of intended test and resource plan
2. Creates HP OpenView map and loads map into HP OpenView

ECS Subsystems

Uses Planning Subsystem tools

DAAC Resource Planner

1. Receives notification to test a data server modification from Test Plan Originator.
2. Assigns Mode Identifier
3. Evaluate Test Plan to identify required resources, View current system resource status using HPOV.
4. Develops a resource plan to implement test.

Computer Operator / Database Administrator

Coordinates resource plan with Resource Manager, Enters resources into planning tool as a ground event for Test.

SMC Resource Manager

SMC is notified of intended plan
Mode Management Support
Points of View II

Computer Operator configures (sets up) resources according to resource plan.

Database Manager sets up a hierarchical-based partition within the databases based on mode identifier.

Computer Operator copies names of support data sets into established structure.

Notify SMC of start of test.

Data Server Subsystem receives modified application and test driver.

SMC Resource Manager

Figure 5.2.5.3-2 Points of View
Mode Management Support
Points of View III

Establishes a new HPOV Session. Initiates Mode Management Service (MMS) from within HP OpenView window.

Selects “Activate Mode” from HPOV GUI. MMS prompts for mode identifier, if non-ops mode it then prompts for simulation time. Resource Manager enters requested information.

Symbols register within the mode specific session and submap for each executable. Operator selects Test Driver symbol and clicks startup.

Uses MSS HP OpenView with custom developed MMS. MSS Agent Service acknowledges new mode and loads mode specific executable info. Test driver executes updated Data Server application. Data Server application(s) register within DCE CDS under specified test mode.
Mode Management Support
Points of View IV

DAAC Resource Manager
- Displays icon(s) on map in test mode window
- Monitors and controls Data Server test application
- HP OpenView detects termination of Data Server application. Icon(s) reflect change of state. Reconfigure system to desired state.
- Notifies SMC of end of test

ECS Subsystems
- Data Server application(s) register with HP OpenView
- Test reads input data from file; writes data to archiver; and updates metadata database. Then it searches metadata database for the same data; loads data from archiver; writes data to output file
- Test executes according to test driver; all data and process interactions isolated to test mode
- After test completes, test driver terminates Data Server application

DAAC Resource Planner

Computer Operator/Database Administrator
- Monitors and controls Data Server test application
- Acknowledges test completion. Reconfigure System to Desired State

SMC Resource Manager
- SMC is notified of end of test