HP UCA Automation

Version 1.1

Integrator's Guide

for Linux RHEL 6.4

Edition: 1.0

October 2014

Legal notices

Warranty

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

License Requirement and U.S. Government Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright notices

Trademark notices

Adobe®, Acrobat® and PostScript® are trademarks of Adobe Systems Incorporated.

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Java™ is a trademark of Oracle and/or its affiliates.

Microsoft®, Internet Explorer, Windows®, Windows Server®, and Windows NT® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Firefox® is a registered trademark of the Mozilla Foundation.

Google Chrome® is a trademark of Google Inc.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

EnterpriseDB® is a registered trademark of EnterpriseDB.

Postgres Plus® is a registered U.S. trademark of EnterpriseDB, Bedford, MA.

UNIX® is a registered trademark of The Open Group.

X/Open® is a registered trademark, and the X device is a trademark of X/Open Company Ltd. in the UK and other countries.

Red Hat® is a registered trademark of the Red Hat Company.

Linux® is a registered trademark of Linus Torvalds in the U.S. and other countries.

Neo4j is a trademark of Neo Technology.

Contents

Figures

Tables

Preface

This guide provides an overview of the UCA Automation product and describes how to create value packs for specific domain specializations and integrate them with the UCA Automation product.

Product Name: UCA Automation

Product Version: 1.1

Read this document before installing or using this software.

Intended audience

This guide is intended for system integrators, solution developers, and software development engineers.

Software versions

The term UNIX is used as a generic reference to the operating system, unless otherwise specified.

The software versions referred to in this document are as follows:

Product Version	**Supported Operating systems**
UCA Automation 1.1	Linux Red Hat Enterprise Linux Server release RHEL 6.4

Typographical conventions

Courier Font:

- Source code and examples of file contents.
- Commands that you enter on the screen.
- Pathnames
- Keyboard key names

Italic Text:

- Filenames, programs and parameters.
- The names of other documents referenced in this manual.

Bold Text:

- To introduce new terms and to emphasize important words.

Reference Documents

- UCA Automation Installation Guide
- UCA Automation Administrator and User Interface Guide
- UCA EBC Problem Detection Installation Administration and Dev Guide
- UCA for Event Based Correlation Value Pack Development Guide
- HP Service Activator Overview Guide
- HP Service Activator PuttingServiceActivatorToWork Guide
- HP Service Activator Plug-ins Guide

Support

You can visit the HP software support online web site at www.hp.com/go/hpsoftwaresupport for contact information, and details about HP Software products, services, and support.

The software support area of the software web site includes the following:

- Downloadable documentation
- Troubleshooting information
- Patches and updates
- Problem reporting
- Training information
- Support program information

Chapter 1

Introduction

The UCA Automation system is a platform for building value added resolution automations based on a combination of business rules and workflow rules. UCA Automation isolates network related issues and automates the corresponding resolutions.

The UCA Automation software is a combination of business rules engine and workflows engine. The system combines HP Unified Correlation Analyzer for Event Based Correlation (UCA EBC) system, which provides the business rules capability with HP Service Activator (HPSA), which provides the activation capability, through the enterprise service bus called NGOSS Open Mediation (NOM).

The following diagram shows the architecture of the UCA Automation system.

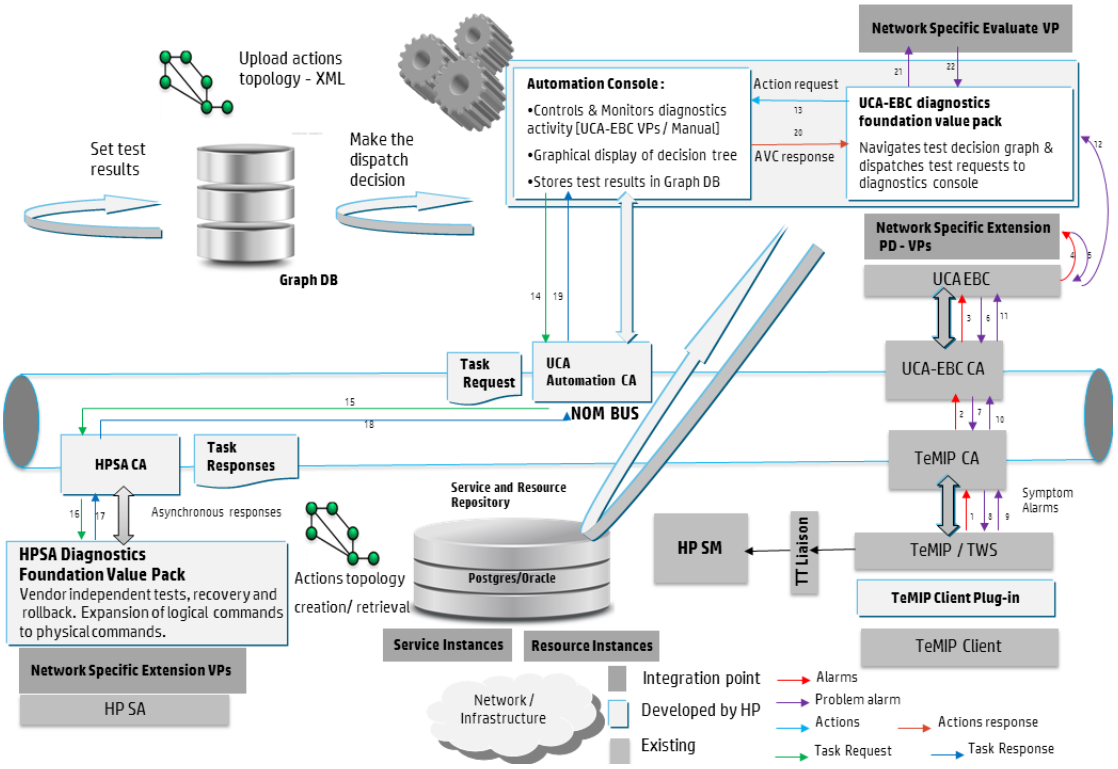

Figure 1 UCA Automation

The grey blocks are the Network Specific Extension Value Packs, optionally Network Specific Evaluate Value Pack, and HPSA Network Specific Extension Value Packs should be implemented to create and integrate custom automations.

1.1 Design theory

Two key functions are performed by UCA Automation; the problem isolation and problem resolution.

Problem isolation is the responsibility of UCA EBC Problem Diagnosis value pack, which can eliminate event storms, false positives, false negatives, and deduce a single meaningful problem alarm.

This information is then passed to the decide-and-act engine, which identifies the action to be taken for a specific problem. After the action, the evolved knowledge is sent back to the decide-and-act engine for further resolution based on the decision tree or evaluate value-pack optionally, to perform predictive and proactive automation. In addition, diagnostic information is gathered automatically to reduce the MTTR (mean time to resolve).

The UCA Automation system works in the way depicted by the following diagram. It starts with the original problem, performs tests after tests as per the decision tree design, and then either resolves the problem or enriches the problem alarm with complete diagnosis, or can even create a trouble ticket automatically.

In case of manual resolution, the operator is presented with a set of problems, the associated services, and a list of the types of devices which can support such services. Once the above triplet is chosen, the corresponding resolutions are displayed, which can be invoked manually.

In UCA Automation System, the process of problem resolution happens in the way depicted by the following diagram. The administrator or integrator of the system has the option to easily configure the decision tree without the need for any kind of programming. The decide and act subsystems work based on this configuration. In case the administrator needs to make advanced decisions based on the results of the previous tests, the platform allows him to write his own rules in the evaluate block.

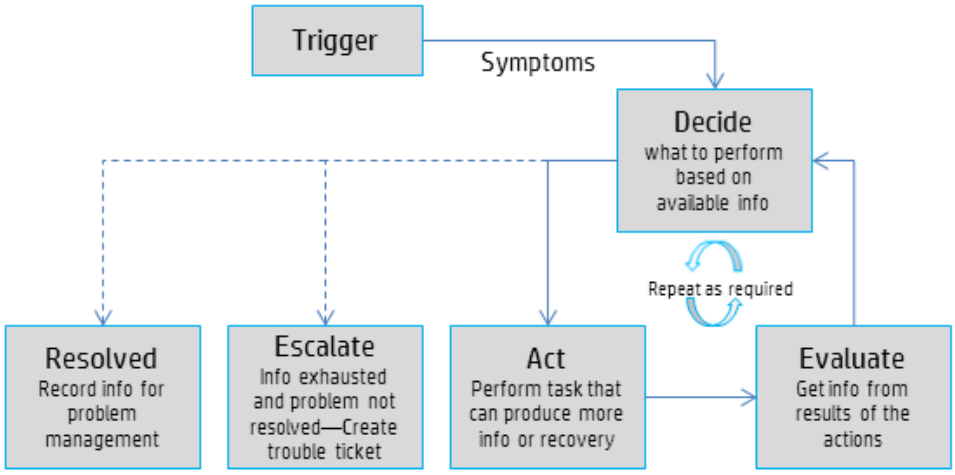

Figure 2 UCA Automation workflow

1.2 Prerequisites

The following are the prerequisites for implementing UCA Automation. Determine the following components before implementing.

- The domain or the service to be automated.

  For example, mobile services, MPLS, ADSL, LTE, ATM, and so on.

  - Identify the service for which the custom automation should be created.

    - Identify network resources which should be associated with these services.
- All problems in that domain and the resolution mechanisms, including:
    - The problem scenario, the characteristics of the root problem, and the filter to be used to isolate this problem.
    - The specific problem/resolution tree for any of the root problems.
    - All resolution actions required for each sub-problem in the problem tree.
    - The input and output parameters for all actions.
    - The method of deducing the output parameters from the raw output using the regex/XML parser.
    - All possible outcomes for the actions.

      Note whether the outcomes are binary or not.
    - Make a differentiation between the primary problems and the results of the actions.
- The decision tree to be built using these problems, actions, and outcomes.

1.3 Implementation

Use the following procedure to implement UCA Automation.

1. Create the domain or service to be automated.

   For example, mobile services, MPLS, ADSL, LTE, ATM, and so on.
    - Create the service according to [R2] chapter 7.
    - Create the network resources according to [R2] chapter 7.
2. Create all the possible problems in that domain and the resolution mechanisms, including the following:
    - A UCA EBC PD value-pack depicting the identified problem scenario with appropriate filters and time-window according to [R3].

      Integrate this value-pack with UCA Automation Foundation value-pack as described in Chapter 2.
    - All resolution actions identified to handle each problem and the outcome of actions. For more details, refer [R2] chapter 7.
    - Appropriate input and output parameters for all actions.
    - Select the appropriate output parameters and their respective parsers.
    - Integrate with the UCA Automation as per Chapter 3.
    - All the primary and secondary problems and associating them with appropriate actions.
3. Create a decision tree with these problems, actions, and outcomes according to the instructions in [R2] Chapter 8.

Chapter 2

Integrate with UCA Automation Foundation value pack

HP UCA Automation Foundation value pack provides the capability to determine the next resolution action based on a reported problem.

The domain specific PD value pack determines and isolates the problem, and delegates the alarm object to the UCA Automation Foundation Value-Pack. After receiving the alarm object with appropriate problem qualification from the network specific PD value pack, the system searches for the resolution in the decision tree available in the Neo4J database and picks an appropriate action.

Based on the action, the foundation value-pack sends out the following XML request to UCA Automaton console for applying the resolution:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<msg xmlns="http://types.ws.ucaautomation.hp.com/">
    <header>
        <ActionRequest Originator="alarm" OpenLoop="true"
Mode="demo">
            <ActionId>100</ActionId>
            <Operation>test_bsc_interface</Operation>
            <OrignatorId>operation_context uca_pbalarm
alarm object 4</OrignatorId>
            <SourceIdentifier>TeMIP EMS</SourceIdentifier>
            <OriginatingManagedEntity>osi_system
site site1</OriginatingManagedEntity>
            <Problem>bsc_interface_down</Problem>
            <ActionPreset>false</ActionPreset>
        </ActionRequest>
    </header>
    <body>
        <Parameters>
            <Parameter>
                <attribute>null</attribute>
                <value></value>
            </Parameter>
            <Parameter>

<attribute>OutputParam:packet_loss</attribute>
                <value></value>
            </Parameter>
        </Parameters>
        <Service>
            <serviceTypeID>MobileServices</serviceTypeID>
            <serviceInstanceID></serviceInstanceID>
        </Service>
        <Resource>
            <resourceTypeID>C3600</resourceTypeID>
```

```
<resourceInstanceID>C3600.ind.hp.com|Ethernet1/0</resourceI
nstanceID>
        </Resource>
    </body>
</msg>
```

2.1 Integrate PD value pack with UCA Foundation pack

After you create the PD value pack with the appropriate problem scenario, the PD value pack provides a complete problem qualification to the UCA Foundation value pack. To provide the problem qualification, use the following procedure.

1. Set the values of the following attributes, for the to be generated Problem Alarm in the calculateProblemAlarmOtherAttribute method.

Attribute	Description
Problem	A predefined Problem Name as defined in the decision tree. The format should be <ServiceType>:<Problem>
Resourceinstance	Resource instance based on how the resource is understood by the activation engine.
Evp	Evaluate Value Pack Name You have to create a network specific evaluate pack if the integrator wants to perform a complex integration and mechanism to determine the next possible problem. After receiving the response for an action, the Foundation value pack intercepts and delegates to a network specific evaluate value pack for further deduction of the problem.
Evpscenario	Evaluate Value Pack scenario The specific scenario to which the system delegates the response.

Table 1 calculateProblemAlarmOtherAttribute attributes

Following is a sample code describing the way to override the calculateProblemAlarmOtherAttribute method.

```
public void calculateProblemAlarmOtherAttribute(Group
group, Action action)
          throws Exception {

   if (LOG.isTraceEnabled()) {
          LogHelper.enter(LOG,
"Problem Site.calculateProblemAlarmOtherAttribute()");
   }
```

```
   action.addCommand("Resourceinstance",
"C3600.ind.hp.com|Ethernet1/0:C3600");
   action.addCommand("Evp","UCA_Automation_DomainExample_UC
A EV");
   action.addCommand("Evpscenario","evaluate");
   action.addCommand("Problem",
"MobileServices:bsc interface down");
   action.addCommand("Resourcetype","c3620");

   if (LOG.isTraceEnabled()) {
          LogHelper.exit(LOG,
"Problem_Site.calculateProblemAlarmOtherAttribute()",action
.toFormattedString());
   }
}
```

The format of the "Resourceinstance" updated in the Problem Alarm must be as follows

<Resource Instance>:<Resource Type>

If the Resource Type is not applicable then specify the value as **None.** The format should be maintained in case the "Resouceinstance" is modified in the Evaluate value pack

The Resource type is optionally used by the workflow's in HPSA Network Specific value pack to parse diagnostic results using regular expressions

2. To provide a complete problem qualification to the UCA Foundation value pack, perform one of the following:

   - (Optional) If the source of the alarms is TeMIP, configure the PD value pack to create the Problem Alarm in a separate Operation Context called `uca_pbalarm`.

     Configure it in the `ProblemXMLConfig.xml` file. Following is snippet of this file.

```
<actions>
   <defaultActionScriptReference>Exec_localhost</defaultAct
ionScriptReference>
   <action name="TeMIP EMS">

   <actionReference>TeMIP AO Directives localhost</actionRe
ference>

   <actionClass>com.hp.uca.expert.vp.pd.actions.TeMIPAction
sFactoryExt</actionClass>

   <attributeUsedForKeyDuringRecognition>userText</attribut
eUsedForKeyDuringRecognition>

   <attributeUsedForKeyPbAlarmCreation>User Text</attribute
UsedForKeyPbAlarmCreation>
          <strings>
                 <string
key="ocName"><value>uca_pbalarm</value></string>
          </strings>
   </action>
</actions>
```

   - If the source of alarms is not TeMIP, delegate the Problem alarm to the UCA Foundation value pack using the delegateEventToScenario() or applyOrchestration() API.

     For more details, refer to the UCA-EBC API documents.

2.2 Integrate Evaluate value pack with UCA Foundation pack

Creating and integrating the network specific evaluate value pack is optional.

Create and integrate the network specific evaluate value pack when you want to interpret the resultant output in very specific ways other than a test passed or test failed criterion.

You can use this value pack to analyze the output from the previous action and can determine the next step or problem to be passed to the foundation value pack. This value pack can contain several scenarios to interpret different outputs from different PD scenarios. It can also contain 1 * n relationships between number of domain specific PD value packs, which represent one scenario each, and evaluate value pack, which represents `n` scenarios.

You should have EBC rules skill to write this value pack. The following snippet shows a scenario where an action response with some parameters is intercepted, the next problem is deduced, and alarm attributes updated in TeMIP are picked up by the foundation value pack for further processing. The output parameters are in the following format

```
packet_loss,100,String:packets_sent,4,String
```

```
rule "Evaluate Action response Rule"
no-loop
 when
 $alarm : EvaluateActionResponse(justInserted == true)
 then
 LogHelper.enter(theScenario.getLogger(),
drools.getRule().getName());

   theScenario.getLogger().trace(Messages.EVALUATE_ACTION_R
ESPONSE_RULE_HAS_FIRED_CORRECTLY.getMessage(new
Object[]{$alarm.toFormattedString()}));
          $alarm.setJustInserted(false);
          $alarm.setScenario(theScenario);
          $alarm.evaluateOutputParams();

   theScenario.getLogger().info(Messages.RETRACTING_THE_ALA
RM_FROM_EVALUATE_ACTION_RESPONSE_RULE.getMessage());
   theScenario.getSession().retract($alarm);

   LogHelper.exit(theScenario.getLogger(),
drools.getRule().getName());
end
```

```
public void evaluateOutputParams()
{
   if (this.getCustomFieldValue("action")
   != null && this.getCustomFieldValue("problem")
   != null &&
   this.getCustomFieldValue("problem").contains("MobileServ
ices"))

{

   if (this.getCustomFieldValue("outputparameters") != "")
{
   String actionOutParams =
this.getCustomFieldValue("outputparameters");

   try {
   this.getDomainProblemInfo();
   this.getActionIdListfromAlarm();
```

```
   if (actionOutParams.contains("packet_loss")) {

   this.evaluatePacketLossParam(actionOutParams);

}
    else if
(actionOutParams.contains("available_interface_name"))

   {
   this.evaluateAvailInterfParam(actionOutParams);
                        }
   else
   {
   LOG.info("Alarm outputParameters doesn't have the
required output parameters to process");

                }
}
 catch (Exception e)
{
   LOG.error("Exception occurred : " + e.getMessage());

   this.getScenario().setStatus("Exception occurred:
"+e.getMessage(),  ScenarioStatus.Degraded);
                }
          }
   else if
   (this.getCustomFieldValue("outputparameters") == "")
   {

   //append the custom attribute actionidlist with
action+" "+actionstatus
   String actionStatus =

   this.getCustomFieldValue("actionstatus");
   LogHelper.method(LOG,
   "EvaluateActionResponse.evaluateOutputParams()"," Alarm
is enriched with new Action Outcome name based on the
action and actionStatus");

   String action = this.getCustomFieldValue("action");
   String actionOutcome = action + "_" + actionStatus;
   String actionIdLists =
   this.getCustomFieldValue("actionidlist");
   this.updateAlarmCustomAttr("Actionidlist",
"actionidlist", actionIdLists+actionOutcome);

   LogHelper.method(LOG,
"EvaluateActionResponse.evaluateOutputParams()"," New
Action Outcome: " + actionOutcome);
          }
   }
}
public void evaluatePacketLossParam(String actionOutParams)
throws Exception {

   List<ActionParameter> actionParameterList =
ParseActionOutParameters
   .parseOutputParameters(actionOutParams);
   //from the outputparameters get the packet loss
   //format packet_loss,100,String
   ActionParameter actionParameter =
```

```
   ParseActionOutParameters
   .getActionParameter(actionParameterList,
Constants.PACKET_LOSS);

   String packetLossValue = actionParameter.getValue();
   //In temip the custom attribute actionidlist is
Actionidlist, but in the working memory it is actionidlist
   //In temip the custom attribute actionstatus is
Actionstatus, but in the working memory it is actionstatus
   if (packetLossValue == null ||
packetLossValue.equalsIgnoreCase("null")
   || packetLossValue.equals("")) {

   updateAlarmCustomAttr("Actionidlist", "actionidlist",
actionIdList+"test_bsc_interface_failed");

   updateAlarmCustomAttr("Actionstatus", "actionstatus",
Constants.STATUS_FAILED);
   }
   else if
   (Integer.parseInt(packetLossValue) > 60)
   {
   //Packetloss is greater than 60, override the
actionstatus from passed to failed

   updateAlarmCustomAttr("Actionidlist", "actionidlist",
actionIdList+"test_bsc_interface_failed");

   if
   (this.getCustomFieldValue("actionstatus").equalsIgnoreCa
se("PASSED"))
   {
   updateAlarmCustomAttr("Actionstatus", "actionstatus",
"FAILED");

   }
   updateAlarmServiceData();
   } else
{
   //Packetloss is less than 60. override the actionstatus
from failed to passed

   updateAlarmCustomAttr("Actionidlist", "actionidlist",
actionIdList+"test_bsc_interface_passed");

   if
   (!this.getCustomFieldValue("actionstatus").equalsIgnoreC
ase("PASSED")) {
   updateAlarmCustomAttr("Actionstatus", "actionstatus",
"FAILED");
         }
   }

}
```

Chapter 3

Integrate with HPSA UCA Automation Controller

The HPSA framework handles the `how` part of the resolution action, which is required for developing new value packs for custom automation. To have an integrated view, the UCA Automation provides a controller workflow with which all the domain specific workflows are integrated.

The task request with the dispatch type as `HPSA` from UCA Automation Console invokes the UCA Controller workflow of the HPSA Foundation Value Pack. Hence, the point of entry for the task request and point of exit for the task response is the UCA Controller workflow. All domain specific workflows are invoked from this workflow.

1. Select **UCA/Parameter** -> **Workflow Templates** view in the HPSA inventory.
2. Create the mapping to the child domain specific workflows.

   Create a mapping of a combination of ServiceType and ActionName with the child domain specific workflow, which is designed to handle such scenarios.

3.1.1 Workflow of a task request

When a task request from the UCA Automation Console invokes the UCA Controller workflow, the Workflow Template is searched automatically to fetch the corresponding child workflow based on the ServiceType and ActionName provided in the task request XML message.

The following snippet shows the Task Request message.

```
<m:msg xmlns:m="http://types.ws.ucaautomation.hp.com/">
   <m:header>
     <m:TaskRequest Mode="real" OpenLoop="true"
Originator="alarm">
       <m:ActionId>100</m:ActionId>
       <m:ActionName>test_bsc_interface</m:ActionName>
       <m:ActionType>test</m:ActionType>
       <m:Operation>Start</m:Operation>
       <m:TaskId>110</m:TaskId>
       <m:OriginatorId>operation context uca pbalarm
alarm_object 4</m:OriginatorId>
       <m:Problem>bsc_interface_down</m:Problem>
     </m:TaskRequest>
   </m:header>
<m:body>
  <m:Parameters>
    <m:Parameter>
    <m:attribute>interface_ip_address</m:attribute>
    <m:value>10.20.30.40</m:value>
  </m:Parameter>
 </m:Parameters>
 <m:Service>
```

```
  <m:serviceTypeID>MobileServices</m:serviceTypeID>
   <m:serviceInstanceID/>
</m:Service>
<m:Resource>
  <m:resourceTypeID>C3600</m:resourceTypeID>
   <m:resourceInstanceID>C3600.ind.hp.com|Ethernet1/0</m:re
sourceInstanceID>
</m:Resource>
</m:body>
</m:msg>
```

Note

You should have controllers for each domain as shown in the Multi Domain Solution image.

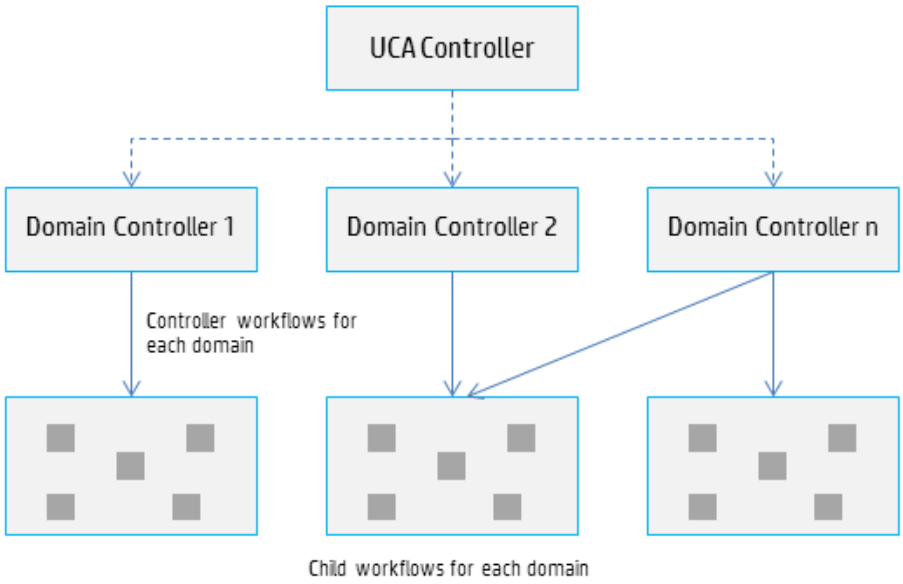

Figure 3 Multi Domain Solution

The following table shows the list of parameters parsed by the UCAController workflow when the domain workflow is invoked. It also shows the expected output case packet variables from the child workflow.

Parameter	Input/Output	Description
message_data	Input	Task request message received from the UCA Automation Console.
problem_name	Input	Name of the problem.
action_name	Input	Name of the diagnostic action.
major_code	Output	Status code for running the child workflow.
minor_code	Output	Status code with information on the execution status of the child workflows.
major_description	Output	Status message of the execution of the child workflow.
minor_description	Output	Status message with information on the execution of the child workflows.

Parameter	Input/Output	Description
Diagnostics	Output	The raw result of the application of the action.
response_string	Output	The output parameter and parsed values are concatenated in a format defined by UCA-EBC. These values are sent as a value in the `outputparameters` tag of the response to UCA–EBC. The format of the string is `<action_name,value,type>,<action_name,value,type>……..<action_name,value,type>`

Table 2 UCAController parameters

The following snippet shows the Task Response message.

```
<resp msg xmlns="http://types.ws.ucaautomation.hp.com/">
    <header>
       <ActionId>100</ActionId>
       <TaskId>100</TaskId>
       <ActionInstanceId>1</ActionInstanceId>
       <OrignatorId>operation_context uca_pbalarm
alarm object 4</OrignatorId>
    </header>
    <body>
        <TaskResponse>
          <MajorCode>
            <Code>501</Code>
            <Description>The test execution failed
            </Description>
          </MajorCode>
          <MinorCode>
            <Code></Code>
            <Description></Description>
          </MinorCode>
            <TaskStatus>FAILED</TaskStatus>
            <Diagnostics>PING 10.20.30.40 (10.20.30.40)
56(84) bytes of data.
       --- 10.20.30.40 ping statistics
       --- 5 packets transmitted, 0 received, 100% packet
loss, time 13999ms
            </Diagnostics>
           <Parameters>
             <Parameter>
               <attribute>outputparameters</attribute>
               <value>packet loss,100,String</value>
             </Parameter>
            </Parameters>
        </TaskResponse>
    </body>
</resp_msg>
```

Following is a snippet of the HPSA Domain Workflow synchronizing with the UCAController.

```
<End-Handler>
    <Name>SyncHandler</Name>
      <Class-Name>
com.hp.ov.activator.mwfm.component.builtin.SyncHandler
      </Class-Name>
      <Param name="job id" value="parent job id"/>
      <Param name="queue"
value="constant:controller_queue"/>
      <Param name="destination0" value="major code"/>
      <Param name="variable0" value="major_code"/>
      <Param name="destination1" value="minor_code"/>
      <Param name="variable1" value="minor_code"/>
```

```
      <Param name="destination2"
value="major_description"/>
      <Param name="variable2" value="major_description"/>
      <Param name="destination3"
value="minor_description"/>
      <Param name="variable3" value="minor_description"/>
      <Param name="destination4" value="diagnostics"/>
      <Param name="variable4" value="raw_result"/>
      <Param name="variable5" value="response_string"/>
   </End-Handler>
```

3.1.2 Status codes

Two sets of status codes are used when implementing domain value packs in HPSA. The status code bundles are located at `${SOLUTION_ETC}/etc/config/messages` in the HPSA Foundation value pack.

- Major code—The major code gives high level information on the execution status and drives the state engine in the UCA Automation Console.

  A sample of major codes and description in messages.properties file is as follows.

```
200=The test was successfully executed
201=The test was partially executed
210=Workflow execution success
300=Request received
400=Bad request, syntax error
401=Invalid request
500=Internal error
501=The test execution failed
```

- Minor code—The minor code and description are secondary codes, which give more information on the status of the execution.

  A sample of minor codes and description in messages.properties file is as follows.

```
402=Parameter: {0} cannot be null/empty
403={0}: {1} was not found in inventory
510={0}: {1} Not Found
511={0} has exceeded the threshold value {1}
512=Free {0} is not available
```

Follow the major code standard according to the HPSA Foundation value pack, as it drives the state engine in the UCA Automation Console.

You should maintain the major and minor code message bundles in a similar way. The domain specific major code message bundle contains all the codes defined in the HPSA Foundation value pack. The minor code message bundle can be defined as per the requirement.

3.1.3 Support for internationalization

The message bundles support internationalization with the help of the custom node. The ResourceBundleReader custom node is available with the Foundation value pack.

Change the file name of the bundle according to the national standards. For example, for French regional setting, the file name is `message_fr.properties`. By default the `message.properties` bundle is picked by the node.

Parameters	Input/Output	Description
bundle_path	Input	The path to the message bundle. In the foundation value pack, the `bundle_path` to the major code messages is `%SOLUTION_ETC%/config/messages/majorcodes`
resource_label	Input	Label of the message bundle. The label in the foundation value pack is `messages`.
Key	Input	Key in the resource bundle. Set the key to `500` if you want the description for this major code.
output_var	Output	Variable in which the fetched string should be stored.
param0..n	Input	This value replaces the constant/variable in the string fetched from the message bundle. `param0` replaces the occurrence of `{0}` in the text. `param1` replaces the occurrence of `{1}` in the text

Table 3 Parameters of the ResourceBundleReader node

3.2 Using HPSA UCA Automation parser

The parser workflow provides a framework for parsing the diagnostic raw result received from the network resources after applying an action.

Both regular expression and Xpath based parsing are supported.

1. Define the parser type when defining an action as the UCA Automation inventory.
2. Create the following directory structure in the solution where `<element_type>` is various types of the device.

```
${SOLUTION ETC}/config/parser/regex/<elementype>/test bsc i
nterface/parser.properties
```

This structure is an example for parsing the output result of a PING action using the regular expression parser.

The parsing information is maintained in the properties files in the `${SOLUTION_ETC}` directory. The properties file contains the mapping of the expected output parameters defined in the inventory for each diagnostic action to its respective regular expression or Xpath expression.

A sample of the `parser.properties` file is as follows.

```
#REGEX mapping for Action: execute_test_on_bsc
#DOMAIN_NAME = com.hp.ov.ucaautomation  (Constant)
#Key -- > DOMAIN_NAME + "." + <ACTION_NAME> + "." +
<PARAMETER>
#ACTION_NAME corresponds to the ACTION_ID of
AUTOMATION_ACTION table in inventory
#Each ACTION_ID has a list of PARAMETERS in the PARAMETERS
table in inventory
#
#All '\' characters in the regex must be escaped for JAVA
#e.g regex pattern for  packetloss
#                       Lost\s=\s\d*\s\((\d*%)\sloss\)  ---
- > Lost\s=\s\d*\s\((\d*%)\sloss\)
```

```
#group id is used to return the input subsequence captured
during the match operation
#Key for group id -- > DOMAIN_NAME + "." + <ACTION_NAME> +
"." + <PARAMETER> + "." + groupid

com.hp.ucaautomation.test_bsc_interface.packet_loss =
(\d*)%\spacket loss,
com.hp.ucaautomation.test_bsc_interface.packet_loss.groupid
= 1
com.hp.ucaautomation.test bsc interface.min time =
\s*Minimum\s=\s(\d*\w*)
com.hp.ucaautomation.test_bsc_interface.min_time.groupid =
1
```

3. Enter the values for the following parameters when invoking the Parser workflow.

Parameter	Input/Output	Description
parser_bundle_label	Input	The name of the parser bundle. In the example in Step 2, the bundle name is `parser`.
parser_bundle_path	Input	The path where parser bundles are available. In the example, the path is `${SOLUTION_ETC}/config/parser/regex/<elementype>/test_bsc_interface`.
parser_type	Input	The type of the parser (regex or xpath). In the previous, the value is `regex`.
action_name	Input	Name of the diagnostic action defined in inventory. In the example, the action name is `test_bsc_interface`.
raw_result	Input	This parameter is the case packet variable which contains the raw information. The raw information is parsed and the data is extracted.
message_data	Input	The request message that was received from the UCA Automation Console.
parameter_map	Output	This map variable contains the mapping of each output parameter of the action to its corresponding parsed values.
minor_code	Output	Status of the workflow execution. A value of `210` represents a successful execution.
minor_description	Output	Diagnostic information of the workflow execution.
response_string	Output	The output parameter and parsed values are concatenated in a format defined by UCA-EBC. This value is sent as a value in the `outputparameters` tag of the response to UCA–EBC. The format of the string is `<action_name,value,type>,<action_name,value,type>……..<action_name,value,type>`

Table 4 Parser parameters

Chapter 4

UCA Automation demo scenario

The UCA Automation kit contains a demo of the automation in the following scenario:

- A series of cell down alarms are generated.
- This storm of alarms is interpreted and the problem is isolated as the BSC is down.
- UCA Automation performs a test to verify whether the BSC is not working or is a false positive.
- If the BSC is down, the system performs a test to check all the available free interfaces.
- Later, the system triggers an action to recover the service, switching it to an available interface.
- After a successful recovery, the alarm is updated with recovery information and all open trouble tickets are closed.
- After a failure from recovery, the alarm is updated with diagnostic information and a trouble ticket is opened.

4.1 Performing the demo scenario

Follow the procedure to run the UCA Automation demo scenario.

1. Deploy the HPSA demo value pack `UCA_HPSA_DomainExample_VP-V11-1A.zip` available under `/opt/UCA_Automation/UCA_Automation_HPSA_VPs` after installation.
2. Deploy the UCA demo value packs `UCA_Automation_DomainExample_UCA_PD-vp-V1.1-1A.zip` and demo evaluate value pack `UCA_Automation_DomainExample_UCA_EV-vp-V1.1-1A.zip`.

   These value packs are available at `/opt/UCA_Automation/UCA_Automation_UCA_VPs`.
3. Database configuration file for the UCA demo value pack.

   Modify the `DBConfiguration.xml` in `/var/opt/UCA-EBC/instances/default/deploy/UCA_Automation_DomainExample_UCA_EV-V1.1-1A/conf`.

   The contents of the file are as follows. Specify the database name, URL, username, and password.

```
#contains the Inventory database access parameters

<DBConfiguration>
    <database>postgres</database>
   <postgressUrl>jdbc:postgresql://localhost:5444/postgres
   </postgressUrl>
    <oracleUrl>
jdbc:oracle:thin:@localhost:1521:hpsadb
```

```
    </oracleUrl>
    <username>hpsa61</username>
     <password>hpsa61</password>
</DBConfiguration>
```

4. Upload the example Decision Tree `/opt/UCA_Automation/Utilities/DecisionTree/etc/DomainEx/DomainEx.xml` using the command line Decision Tree utility

5. Edit the ${UCA_EBC_INSTANCES}/conf/OrchestraConfiguration.xml file and add the following route configuration

```
<Routes>
   <Route name="Copy from UCA Automation Foundation VP to
from UCA Automation EV VP ">
    <COPY>
     <Source>
   <ValuePackNameVersion>UCA_Automation_Foundation_UCA-
V1.1-1A</ValuePackNameVersion>

   <ScenarioName>UCA_Automation_Foundation_UCA.requestrespo
nse</ScenarioName>
   </Source>
   <Destinations>
      <Destination>
          <Target>

<ValuePackNameVersion>UCA_Automation_DomainExample_UCA_EV-
V1.1-1A</ValuePackNameVersion>

<ScenarioName>UCA_Automation_DomainExample_UCA_EV.evaluate<
/ScenarioName>
               </Target>
          </Destination>
   </Destinations>
   </COPY>
          </Route>
   </Routes>
```

6. Generate alarms from TeMIP, which match the pattern present inside the PD value pack `UCA_Automation_DomainExample_UCA_PD-vp-V1.1-1A.zip`. Ensure that the TeMIP operation contexts uca_network and uca_pbalarm are created. Refer the UCA Automation Installation guide for more details. Run the sample alarm generation utility script provided with the Domain example PD value pack in the bin folder.

7. Interact with the Tasks from the UCA Automation console.