HP UCA Automation

UCA Automation

Version 1.0

Integrator's Guide

Edition: 1.1

For the Operating Systems:
Linux (RHEL 6.4)

November 2013

© Copyright 2013 Hewlett-Packard Development Company, L.P.

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

© Copyright 2013 Hewlett-Packard Development Company, L.P.

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

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 UC Automation product.

Product Name: UCA Automation

Product Version: 1.0

Kit Version: V1.0

Please read this document before installing or using this Software.

Intended Audience

Here are some recommendations based on possible reader profiles:

- System Integrators
- Solution Developers
- 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 1.0	Linux Red Hat Enterprise Linux Server release 6.4

Table 1 - Software versions

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

- [R1] HP UCA Automation V1.0 – Installation Guide V1.1
- [R2] HP UCA Automation V1.0 - Administrator and User Interface Guide V1.1
- [R3] HP UCA EBC Problem Detection V3.0 - Installation Administration and Dev Guide V1.0
- [R4] HP UCA for Event Based Correlation V3.0 - Value Pack Development Guide V1.0
- [R5] HP Service Activator Overview.pdf
- [R6] HP Service Activator PuttingServiceActivatorToWork.pdf
- [R7] HP Service Activator Plug-ins.pdf
-

Support

Please visit our 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

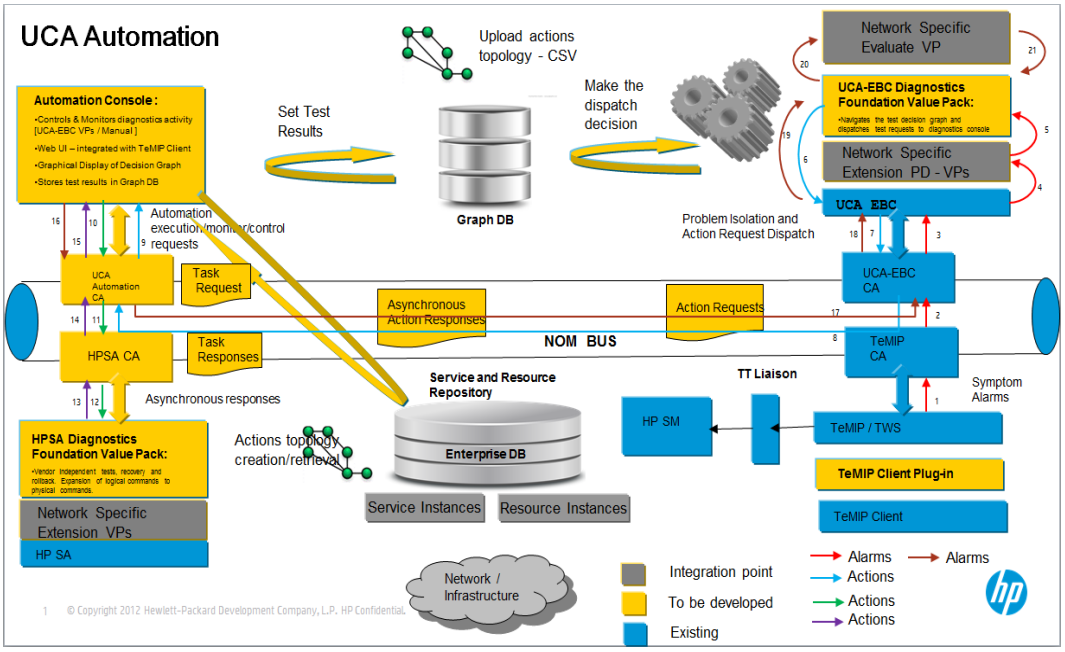
The objective of this document is to provide details on the different interfaces provided by UCA Automation – to develop a custom automation, to describe and recommend the general methodology to be used while developing and integrating custom automation. The following sections of this chapter introduce the user to the purpose of UCA Automation, and then describe the architecture in brief and finally to describe the concepts and methodology behind developing and integrating a custom automation.

In any typical service provider environment, there's always a need for isolation of network related issues and automated resolutions of the same. UCA Automation Software is positioned primarily to address this need. It is implemented as a combination of business rules engine and workflows engine. The system would involve the integration of HP Unified Correlation Analyzer for Event Based Correlation [UCA EBC] system [which provides business rules capability] and HP Service Activator [HPSA] [which provides activation capability] glued via the enterprise service bus called NOM [NGOSS Open Mediation].

Generally, in the resolution solutions available today, there's no separate layering between '***what resolution steps***' need to be carried-out upon the incidence of a specific issue and '***how these resolution steps are carried out***'. This mixed up implementation of processes which represent both ***what*** and ***how*** part of the logic on the top of workflow engines OR in some cases on top of business rules engines, makes the workflows or business rules very complex to develop, comprehend, debug and maintain [read as '***modify'*** when a business decision changes – say support a new device type, support a new resolution command on the same device or support a new format of the output for an existing resolution command with an upgrade in device firmware].The problem would scale to unimaginable magnitudes considering the different technologies such as DWDM, SDH, DSL, MPLS, LTE [and legacies such as ATM, FR and X.25] and different layers / types of networks such as 'the transport', 'the access', 'the core' , 'the radio access' and so on.

UCA Automation Software which is a combination of both business rule engine and the workflow engine will enable a clear separation of **what to automate** and **how to automate**. All the complexities of actual automation such as how to access a network resource *[could* be a network element, an element component or an EMS or *NMS]*, what it's credentials could be, which specific transport mechanism to use to connect to the resource, what specific OS version of the device are to be supported, what specific commands need to be sent, would be abstracted from the business rules. This would enable the administrators to **create-update-read** the business rules with **utmost clarity** and **maintain** them **efficiently**. This would empower the administrators to store the knowledge gained regarding the automation in the form of business rules focusing on **what part** without bothering about the **how part**. One another advantage of UCA Automation software is, for most of the resolution automations – it would require the operator only to know business rules and he need not have knowledge of the business rules technologies to implement day to day operational changes to the decisions.

Thus UCA Automation System is a platform for building value added resolution automations based on a judicious combination of business rules and workflows. Following diagram shows the overall architecture of UCA Automation System.



From the above architectural diagram – clearly, the integration interfaces for customized automations fall under the following category. The grey blocks 'Network Specific Extension Value Packs', optionally 'Network Specific Evaluate Value Packs', 'HPSA Network Specific Extension value packs' need to be implemented - to create and integrate custom automations.

Integration with UCA Automation foundation value pack

Integration with UCA Automation evaluate value pack

Integration with HPSA UCA Automation Controller workflow

Decision tree for automation

1.1 Design thoughts behind UCA Automation based resolution

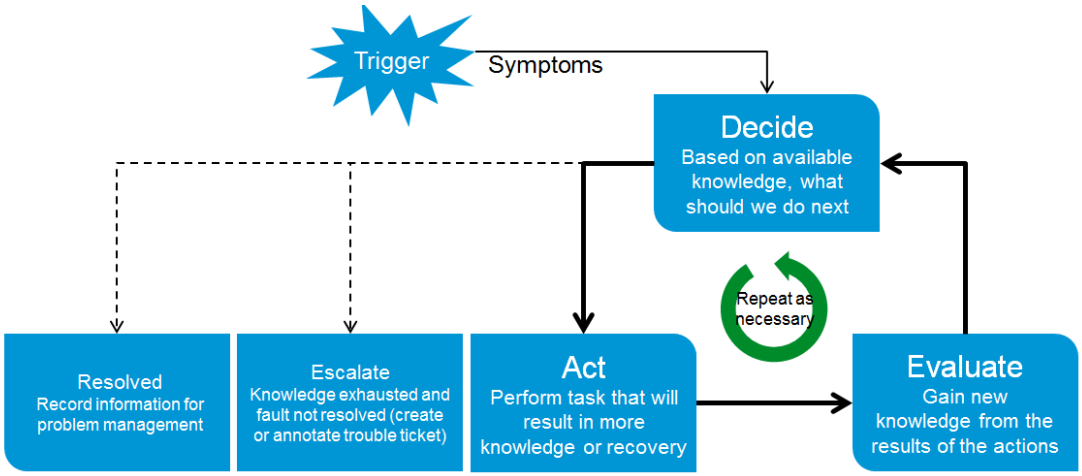
The two key functionalities but, not limited to - performed by UCA Automation are - problem isolation and problem resolution. Problem isolation is the responsibility of UCA EBC Problem Diagnosis value pack – which has the capability to eliminate event storms, false positives and false negatives and deduce one meaningful single "problem alarm" instead of hundreds. This information is then passed on to the "decide and act" engine – which identifies what action needs to be taken for a specific problem. Once the action is executed, the evolved knowledge / status are sent back to "decide and act" engine for further resolution according to the decision tree or "evaluate value pack" optionally, to perform predictive, proactive automation. In addition lots of diagnostic information is gathered automatically to reduce the MTTR [mean time to resolve].

UCA Automation system would work in the way depicted by the following diagram, starts with the original problem, performs tests after tests as per the decision tree design and would either resolve the problem or enrich the problem alarm with complete diagnosis of all the steps performed and can even create a trouble ticket automatically.

In case of manual resolution, the operator is presented a set of problems, the services they are

associated with and the list of the types of devices which could 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.



1.2 Ground work needed before the implementation begins

- Identification of domain / service to be automated [e.g. mobile services / MPLS/ ADSL, LTE, ATM etc.]
    - Identify the service for which custom automation needs to be created
    - Identify the network resources which are to be associated with these services
- Identification of all the problems in that domain and the resolution mechanisms
    - Identify the problem scenario, the root cause problem characteristics, and the filter to be used to isolate this problem.
    - Identify the specific problem / resolution tree for any of the root cause problems
    - Identify all the resolution actions to be taken for each of the sub-problems in the problem tree.
    - Identify the input and output parameters for all the actions
    - Identify how the output parameters must be deduced from the raw-output using the regex / xml parser

    - Identify all the possible outcomes for the actions; keep in mind whether the outcomes are binary or n-ary.
    - Make a differentiation between the “primary problems” – and the results of the actions.
- Identify the decision tree to be built using the above set of problems, actions and outcomes.

1.3 Implementation

- Creation of domain / service to be automated [e.g. mobile services / MPLS/ ADSL, LTE, ATM etc.]
    - Creation of the service as per [R2] chapter 7.
    - Creation of the network resources as per the [R2] chapter 7
- Creation of all the problems in that domain and the resolution mechanisms
    - Create a UCA EBC PD Value pack depicting the identified problem scenario, with appropriate filters and time-window – as per [R3]. This needs to be integrated with UCA Automation Foundation value pack as described in chapter 2.
    - Create all the resolution actions identified to handle each of the primary problems and the outcomes of actions [as per [R2] chapter 7]
    - Create appropriate input and output parameters for all the actions
    - Choose the appropriate output parameters and their respective parsers. This needs to be integrated with the UCA Automation as per the chapter 3.
    - Create all the primary and secondary problems and associate them with appropriate actions.
- Create the decision tree with the above set of problems and actions as per the instructions given in [R2] chapter 8.

Chapter 2

Integration with UCA Automation Foundation Value Pack

HP UCA Automation Foundation value pack performs the functionality of determining the next resolution action based on the problem passed on to it. It's the responsibility of the domain specific PD value pack to determine and isolate the problem and delegate the alarm object to the UCA Automation Foundation Value Pack. Once an alarm object with appropriate problem qualification is received from network specific PD value pack, a look up in the decision tree present in Neo4J database would be performed and appropriate action is picked up. Up on this the foundation value pack would send out the following XML request to UCA Automaton console for the execution of resolution action:

```
<m:msg xmlns:m="http://types.ws.ucaautomation.hp.com/">
    <m:header>
        <m:TaskRequest Mode="demo" 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>100</m:TaskId>
            <m:OriginatorId>operation_context uca_network alarm_object 1958</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>127.0.0.1</m:value>
            </m:Parameter>
        </m:Parameters>
        <m:Service>
            <m:serviceTypeID>MobileServices</m:serviceTypeID>
            <m:serviceInstanceID/>
        </m:Service>
        <m:Resource>
            <m:resourceTypeID>c3620</m:resourceTypeID>
            <m:resourceInstanceID>C3600.ind.hp.com|Ethernet1/0</m:resourceInstanceID>
        </m:Resource>
    </m:body>
</m:msg>
```

2.1 Integration of PD value pack with UCA Foundation pack

Once the PD value pack representing the appropriate problem scenario is designed, complete problem qualification needs to be provided by the PD value pack to the UCA foundation value pack. In order to perform this, the generated method *'whatToDoWhenProblemAlarmIsAttachedToGroup'* - needs to be overridden to populate various user defined attributes of alarm objects such as :

- Problem Name [predefined]
- Resource instance [as the resource is understood by the activation engine]
- Evaluate Value Pack Name [In case the integrator would like to perform complex integration and complex mechanism to determine the next possible problem – a network specific evaluate pack needs to be created and this needs to be provided here]. Once the response for an action is received, it would be intercepted by the foundation value pack and would be delegate*d 'network specific evaluate value pack' – for further deduction of the problem.
- Evaluate Value Pack scenario [The specific scenario to which the response needs to be delegated to].

Following is an example code snippet describing the way to override the method 'whatToDoWhenProblemAlarmIsAttachedToGroup'.

```
public void whatToDoWhenProblemAlarmIsAttachedToGroup(Group group)
            throws Exception {
            if (LOG.isTraceEnabled()) {
                  LogHelper.enter(LOG, "Problem_Site.whatToDoWhenProblemAlarmIsAttachedToGroup()");
            }
            super.whatToDoWhenProblemAlarmIsAttachedToGroup(group);
            recomputeAdditionalText(group, true);

            synchronized (group.getAlarmsMap()) {
                  for (Alarm alarm : group.getAlarmList()) {
                        getScenario().getGroups().setSetAs(group, alarm,
                                    Qualifier.SubAlarm);
                  }
            }
            Alarm problemAlarm = group.getProblemAlarm();
            if (LOG.isInfoEnabled()) {
                  LOG.info(Messages.ENRICHING_THE_PROBLEM_ALARM_WITH_REQUIRED_INFORMATION
                              .getMessage(new Object[] { PROBLEM_NAME, RESOURCE_INSTANCE_VALUE, EVALUATE_VALUEPACK_NAME,
                                          EVALUATE_VALUEPACK_SCENARIO }));
            }
            //Setting resource instance in problem alarm
            updateAlarmCustomAttr(problemAlarm, CF_RESOURCE_INSTANCE, RESOURCE_INSTANCE_VALUE);
            //Setting evaluate value pack name in problem alarm
            updateAlarmCustomAttr(problemAlarm, CF_EVP, EVALUATE_VALUEPACK_NAME);
            //Setting evaluate value pack scenario name in problem alarm
            updateAlarmCustomAttr(problemAlarm, CF_EVP_SCENARIO, EVALUATE_VALUEPACK_SCENARIO);
            //Set the problem in problem alarm
            updateAlarmCustomAttr(problemAlarm, CF_PROBLEM, PROBLEM_NAME);
            if (LOG.isInfoEnabled()) {
                  LOG.info(Messages.DELEGATING_ALARM_TO_FOUNDATION_VP
                              .getMessage(new Object[] { FOUNDATION_VALUEPACK_NAME,
                                          FOUNDATION_VALUEPACK_SCENARIO }));
            }
            if (LOG.isTraceEnabled()) {
                  LogHelper.method(LOG, "The problem alarm delegated from PDVP is: " + problemAlarm.toFormattedString());
            }
            getScenario()
                        .delegateAlarmToScenario(
                                    FOUNDATION_VALUEPACK_NAME,
                                    VALUEPACK_VERSION,
                                    FOUNDATION_VALUEPACK_NAME + "."
                                                + FOUNDATION_VALUEPACK_SCENARIO, problemAlarm);
            if (LOG.isTraceEnabled()) {
                  LogHelper.exit(LOG, "Problem_Site.whatToDoWhenProblemAlarmIsAttachedToGroup()");
            }
      }
```

2.2 Integration of EVALUATE value pack with UCA Foundation pack

The very creation and integration of ‘network specific evaluate’ value pack is optional. The integrator is recommended to use this in cases where the resultant output needs to be interpreted in very specific ways other than a ‘test passed’ or ‘test failed’ paradigm. It could be used to analyze the output from the previous action and can deduce what could be the next step / problem to be passed on to the foundation value pack. This value pack can have several scenarios defined – to interpret different outputs from different PD scenarios. Mathematically there could be 1Xn relationship between number of domain specific PD value pack which represent one scenario each and evaluate value pack which represents ‘n’ scenarios. Note that, writing this value pack would require UCA EBC rules skill. Following is a code snippet demonstrating a scenario where an action response with some parameters is being intercepted and next problem is deduced and delegated to the foundation value pack for further processing:

```
rule "Evaluate Action response with output parameters"
no-loop
	when
	 $alarm : EvaluateActionResponse(getCustomFieldValue(Constants.ACTION) != null && getCustomFieldValue(Constants.ACTION_OUT_PARAMS) != null
			&& getCustomFieldValue(Constants.PROBLEM) contains Constants.DOMAIN)
	 then
		theScenario.getLogger().info(Messages.EVALUATE_ACTION_RESPONSE_RULE_HAS_FIRED_CORRECTLY.getMessage(new Object[]{$alarm.toFormattedString()}));
			String actionOutParams = $alarm.getCustomFieldValue(Constants.ACTION_OUT_PARAMS);
			theScenario.getLogger().trace(" Alarm actionOutParams: " + actionOutParams);
			$alarm.setScenario(theScenario);
			try {
				$alarm.getDomainProblemInfo();

			if (actionOutParams.contains(Constants.PACKET_LOSS)) {
					$alarm.evaluatePacketLossParam(actionOutParams);
				} else if (actionOutParams.contains(Constants.AVAILABLE_INTERFACE_LIST)) {
				$alarm.evaluateAvailInterfParam(actionOutParams);
				} else {
					theScenario.getLogger().info("Alarm outputParameters doesn't have the required output parameters to process");
				}
			theScenario.getLogger().info(Messages.DELEGATING_ALARM_TO_FOUNDATION_VP.getMessage(new Object[] {Constants.FOUNDATION_VALUEPACK_NAME, Constants.FOUNDATION_VALUEPACK_SCENARIO}));
			theScenario.getLogger().trace("Delegated alarm is:" + $alarm.toFormattedString());
			theScenario.delegateAlarmToScenario(Constants.FOUNDATION_VALUEPACK_NAME, Constants.VALUEPACK_VERSION, Constants.FOUNDATION_VALUEPACK_NAME +"."+ Constants.FOUNDATION_VALUEPACK_SCENARIO,
$alarm);
		} catch (Exception e) {
			theScenario.getLogger().error("Exception occured : " + e.getMessage());
			theScenario.setStatus("Exception occured: "+e.getMessage(),  ScenarioStatus.Degraded);
		}
		theScenario.getLogger().info(Messages.RETRACTING_THE_ALARM_FROM_EVALUATE_ACTION_RESPONSE_RULE.getMessage());
		theScenario.getSession().retract($alarm);
		Flag endFlag=new Flag(AbstractJunitIntegrationTest.TEST_END_FLAG_ID ,"this is the end of the rule", true);
		theScenario.getSession().insert(endFlag);
	end
```

Chapter 3

Integration with HPSA UCA Automation Controller

The custom automation needs new value packs to be developed using HPSA framework, which would handle the ‘how’ part of the resolution action. To have an integrated view UCA Automation provide an controller work flow with which all the domain specific work flows needs to be integrated. The following two sections in this chapter talk about the integration with UCA Automation controller workflow and the integration with parser framework provided by UCA Automation.

3.1 Integration with HPSA UCA Automation Controller

As task request (with dispatch type as HPSA) from UCA Automation Console invokes the UCAController workflow of the HPSA Foundation Value Pack. So the point of entry for the task request and point of exit for the task response is the UCAController workflow. All domain specific workflows will be invoked from this workflow.

In the HPSA inventory the mapping to the child domain specific workflows can be created from the UCA/Parameter→Workflow Templates view. As an integrator a mapping of combination of ServiceType, Problem and ActionName must be created with the child domain specific workflow which is designed to handle such scenarios. When a task request from the UCA Automation Console invokes the UCAController workflow, the WorkFlow Template is looked up automatically to fetch the corresponding child workflow based on the ServiceType, Problem and ActionName present in the task request xml message.

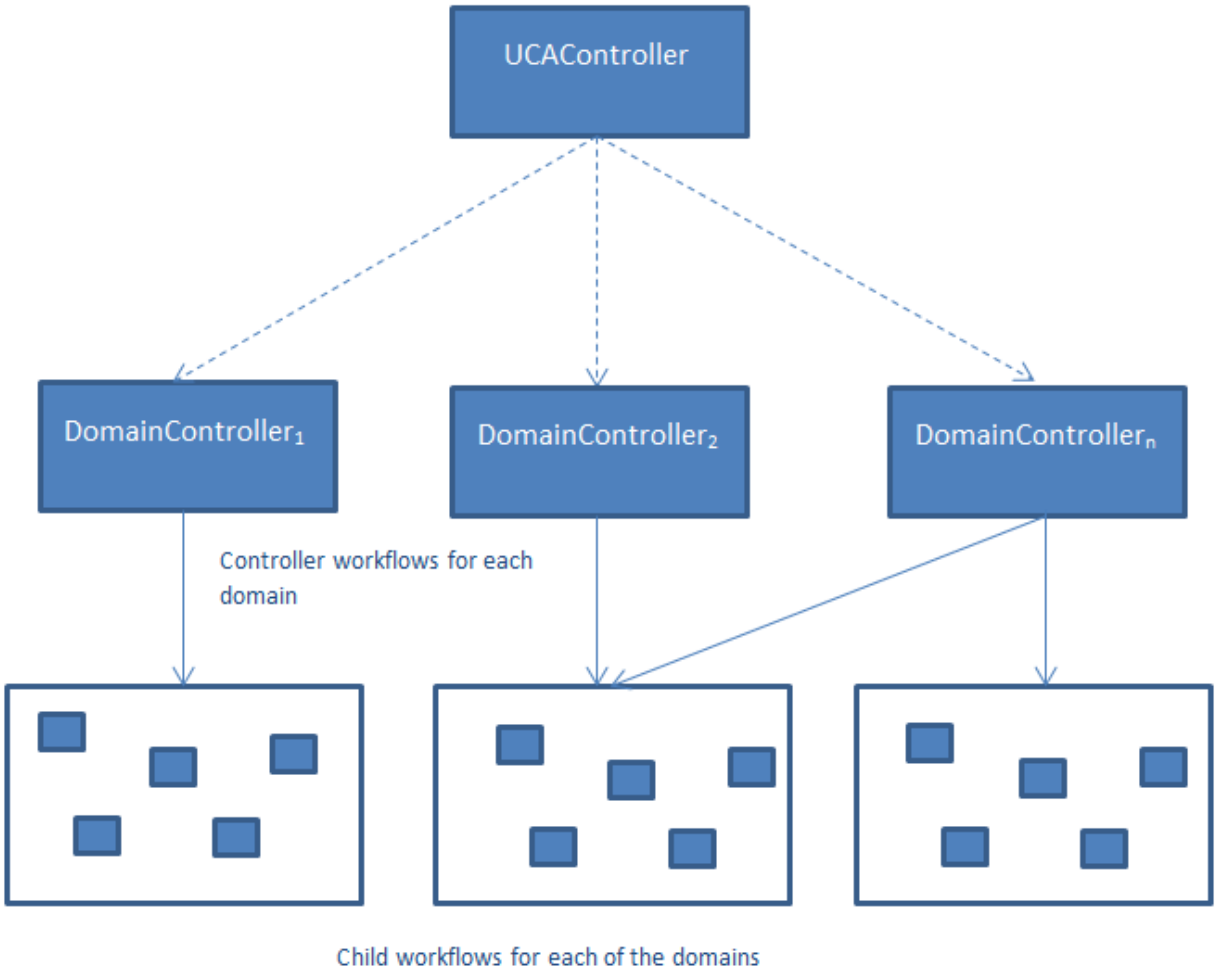


Illustration of a multi domain solution

It is recommended to have controllers for each of the domains as shown in the figure above.

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

Parameter	Input/Output	Description
message_data	Input	The task request message received from the UCA Automation Console
problem_name	Input	The name of the problem
action_name	Input	The name of the diagnostic action
major_code	Output	Status code of the execution of the child workflow.
minor_code	Output	Status code providing further 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 providing further information on the execution of the child workflows
Diagnostics	Output	The raw result output of the execution of the action
response_string	Output	The output param and parsed values are concatenated in a format defined by UCA-EBC. This 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>

The major code standard has to be followed as per the HPSA Foundation value pack since this is used to drive the state engine in the UCA Automation Console. There are two set of status code that is to be used while implementing the domain value packs in HPSA. The major code gives high level information on the execution status and drives the state engine in the UCA Automation Console. The major description is the textual meaning of the code. The minor code and description are secondary codes and give more information on the status of the execution.

The status code bundles are located at ${SOLUTION_ETC}/ etc/config/messages in the HPSA Foundation value pack. A sample of the codes is shown below.

```
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
```

Major codes and description in messages.properties

```
402=Paramter: {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
```

Minor codes and description in messages.properries

The implementer of the HPSA domain value pack is recommended to maintain the major and minor code message bundles in a similar fashion. The domain specific major code message bundle should contain all the codes defined in the HPSA Foundation Value pack since these codes are used to drive the state engine in the UCA Automation Console. The minor code message bundle can be defined as per their requirement.

The message bundles support internationalization with the help of the custom node. The ResourceBundleReader custom node is included along with the Foundation value pack. The file name of the bundle has to be changed according to the internatiolization standards, e.g for French regional setting the file name would be message_fr.properties. By default the message.proeprties bundle will be 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	The label of the message bundle. The label in the foundation value pack is messages
Key	Input	The key in the resource bundle. The key would be 500 is we want the description for this major code
output_var	Output	The variable in which the fetched string is to be stored
param0..n	Input	This replaces the constant/variable into the string fetched from the message bundle, param0 will replace occurrence of {0} in the text, param1 will replace occurrence of {1} in the text

Parameters of the ResourceBundleReader Node

3.2 Integration with HPSA UCA Automation Parser

The parser workflow provides a framework for parsing the diagnostic (raw result) received from the network resources on execution of an action. Both regular expression and Xpath based parsing are supported. When defining an action is the UCA Automation inventory the parser type can be defined.

The parsing information should be maintained in properties files within the ${SOLUTION_ETC} directory. The properties file should contain the mapping of the expected output parameters (already defined in the inventory for each diagnostic action) to its respective regular expression or Xpath expressions.

As an example,

Let us consider that the output result of a PING action is to be parsed using the regular expression parser. We can create the following directory structure in the solution where <element_type> can be used to distinguish between various types of the device.

${SOLUTION_ETC}/config/parser/regex/<elementype>/test_bsc_interface/parser.properties.

The parser.properties file is shown below

```
#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
```

When the invoking the Parser workflow the following parameters are mandatory.

Parameter	Input/Output	Description
parser_bundle_label	Input	The name of the parser bundle. From the above example the bundle name is *parser*
parser_bundle_path	Input	The path where parser bundles are available. From the above example the path is ${SOLUTION_ETC}/config/parser/regex/<elementype>/test_bsc_interface
parser_type	Input	The type of the parser (regex|xpath). From the above example it is regex
action_name	Input	Name of the diagnostic action defined in inventory. From the above example the action name is test_bsc_interface
raw_result	Input	This is the case packet variable which contains the raw information which needs to be parsed and the data extracted
message_data	Input	The actual request message that was received from the UCA Automation Console
parameter_map	Output	This map variable contains the mapping of each of the output parameters of the action to its corresponding parsed values
minor_code	Output	Status of the workflow execution. 210 represents a successful execution
minor_description	Output	Diagnostic information of the workflow execution
response_string	Output	The output param and parsed values are concatenated in a format defined bu UCA-EBC. This 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>

Chapter 4

Using the demo automation scenario

The UCA Automation kit comes with demo automation depicting the following scenario:

- A series of cell down alarms would be generated
- This storm of alarms is interpreted and the problem is isolated to be BSC down alarm
- UCA Automation performs a test to see if BSC is really down or it's a false positive
- In case the BSC is down, a test is performed to check all the available free interfaces
- Then an action is triggered to recover the service switching it to an available interface
- Upon successful recovery the alarm is updated with recovery information and any open trouble ticket would be closed
- Upon failure of recovery, alarm would be updated with diagnostic information and a trouble ticket is opened.

4.1 Seeing the demo work

Perform the following steps to see the UCA Automation demo – working:

- Deploy the HPSA demo value pack UCA_HPSA_DomainExample_VP-X10-1A.zip present in /opt/UCA_Automation/ UCA_Automation_HPSA_VPs after installation. Refer [1]
- Deploy the UCA demo value packs UCA_Automation_DomainExample_UCA_PD-vp-V1.0-1A.zip and demo evaluate value pack UCA_Automation_DomainExample_UCA_EV-vp-V1.0-1A.zip, as per [1]. These value packs would be available at /opt/UCA_Automation/UCA_Automation_UCA_VPs.
    - Database configuration file for the UCA demo value pack. Create a file MobileServices_Config.properties in /var/opt/UCA-EBC/instances/default/config. The contents of the file should be as follows

```
#contains the Inventory database access parameters

#Set jdbc.database as oracle or postgres.
jdbc.database=postgres
#Default <Postgres DB Port>is 5444 and <Oracle  DB Port> is 1521
```

jdbc.postgress.url=jdbc:postgresql://<DB server>:<Postgres DB Port>/<DB schema>
jdbc.oracle.url=jdbc:oracle:thin:@ <DB server>:<Oracle DB Port>:<DB schema>

jdbc.username=<DB User>
jdbc.password=<DB password>

- Generate alarms from TeMIP – which are matching the pattern present inside the PD value pack UCA_Automation_DomainExample_UCA_PD-vp-V1.0-1A.zip.