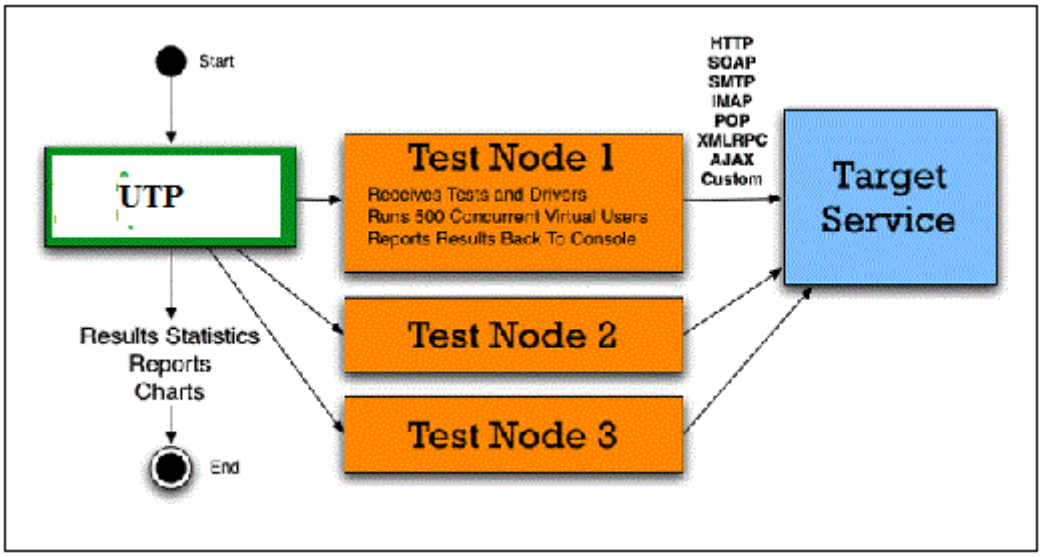
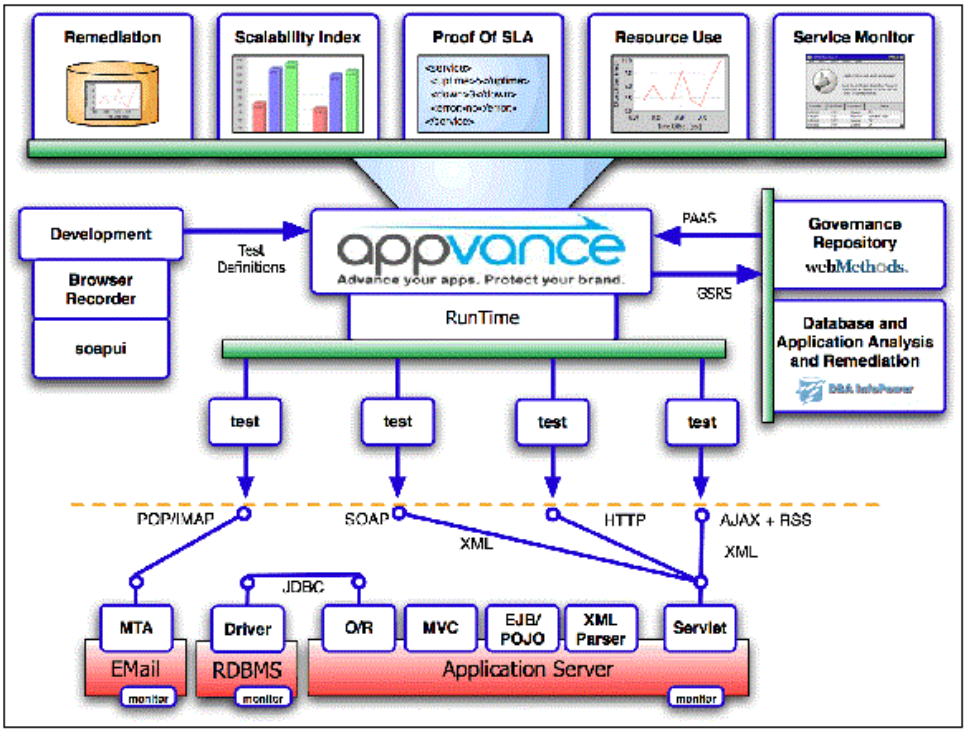
The PAAS Interface

Appvance IQ is an automation framework with a distributed test environment to test services for functionality, scalability, and performance. The AIQ Console sends required information to specialized and optimized TestNodes (located on several computers), runs the tests, and sends results back to the Console. The Console processes the results to create statistics files, charts, and reports. A simplified workflow of AIQ.



Appvance IQ enables the Console to receive soap-based Web service requests to operate a Test Scenario. AIQ has been upgraded to allow any application using a connector class to connect to the Console and run a test, with Appvance As A Service. The upgraded work-flow of AIQ;



Modify the TestNodes to Allow Them to Operate Multiple Tests Concurrently.

In previous versions of the Appvance IQ TestNetwork architecture, TestNodes could receive instructions from only one Console. The new Appvance IQ architecture allows TestNodes to receive instructions from multiple applications (Consoles), which have been checked for availability, stability, and confidentiality.

1. Availability means a scenario can access all of its resources
   To accomplish this, each scenario is given a unique ID in the form of ID_ plus an ID number. Using the ID as its name, a directory (the application's root directory) is created and contains all the resources of each test
2. Stability means any process can not affect any other process.
   To accomplish this, each monitor is run as an isolated thread with its own variable and ClassLoader, used to load the new required jar resources for a JAVA test
3. Confidentiality means any scenario can not access any other scenario's data.
   To accomplish this, each scenario only knows its own root directory

Develop a Class that Becomes a Service

1. The goal was to create a new class and use Apache Axis to turn it into a Web Service. This involved two tasks, compiling the new class and configuring tomcat to accept the new service
2. The new class needed to be simple, yet functional and complete as described in Classes

Classes

Function Name	Parameter	Explanation
RunTest	String aFilenames	Run the test and get its session ID. NOTE: This is required for the other functions.
	DataHandler aDHFiles	First argument is the scenario's required filenames. Second argument is the datahandlers containing the files' information. This argument is the test scenario's filename. Fourth and firth arguments are the username and password used for authentication.
	String sScript String user String pswd	Returns an integer representing the session ID. Error if zero; system was not able to create scenario.
TestStatus	int id String user String pswd	Get the test's status. First argument is the session ID (obtained using the Runtest function). Second and third arguments are the username and password for authentication. Returns a single value indicating test status: 0 - Test operating normally. 1 - Test running and encountered one or more exceptional conditions. 2 - Test paused. 3 - Test ended. 4 - Error; current errors failed authentication, or session ID is not valid (iD<=0, ID for non-existent test).
GetSnapshot	int id String user String pswd	Get test's most recent result. First argument is the session ID (obtained using RunTest function). Second and third arguments are the username and password for authentication. Returns a GSRS formatted XML document with the test's most recent result. Error if NULL (ID is not valid or authetication failed).
PauseTest	int id String user String pswd	Pause the test. First argument is the session ID (obtained using the RunTest function). Second and third arguments are the username and password for authentication. Returns true when test paused successfully (even if already paused). Error if false (authentication failed or ID is not valid).
ResumeTest	int id String user String pswd	Resume a paused test. First argument is the session ID (obtained using the Runtest function). Second and third arguments are the username and password for authentication. Returns true when test resumes successfully (even if already running). Error if false (authentication failed or ID is not valid).
StopTest	int id String user String pswd	Stop a test. First argument is the session ID (obtained using RunTest function). Second and third arguments are the username and password for authentication. Returns true when test stops successfully. Error if false (test already stopped, authentication failed, ID is not valid).

Configuring Axis to accept the new class as a new service was surprisingly easy; simply add a new service tag to the Axis configuration file (server-config.wsdd)

```
<service name="PAAS" provider="java:RPC">
<parameter name="allowedMethods" value="*"/>
<parameter name="className" value="com.Appvance.testnetwork.PAAS"/>
</service>
The new service is located on the server at:
http://ServerName:8080/TestNetwork/ws/PAAS
The requests in the service WSDL definition are:

<wsdl:message name="RunTestRequest">
<wsdl:part name="in0" type="impl:ArrayOf_xsd_string"/>
<wsdl:part name="in1" type="impl:ArrayOf_apachesoap_DataHandler"/>
<wsdl:part name="in2" type="xsd:string"/>
<wsdl:part name="in3" type="xsd:string"/>
<wsdl:part name="in4" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="TestStatusRequest">
<wsdl:part name="in0" type="xsd:int"/>
<wsdl:part name="in1" type="xsd:string"/>
<wsdl:part name="in2" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="GetSnapshotRequest">
<wsdl:part name="in0" type="xsd:int"/>
<wsdl:part name="in1" type="xsd:string"/>
<wsdl:part name="in2" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="PauseTestRequest">
<wsdl:part name="in0" type="xsd:int"/>
<wsdl:part name="in1" type="xsd:string"/>
<wsdl:part name="in2" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="ResumeTestRequest">
<wsdl:part name="in0" type="xsd:int"/>
<wsdl:part name="in1" type="xsd:string"/>
<wsdl:part name="in2" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="StopTestRequest">
<wsdl:part name="in0" type="xsd:int"/>
<wsdl:part name="in1" type="xsd:string"/>
<wsdl:part name="in2" type="xsd:string"/>
</wsdl:message>

The responses in the service's WSDL are:

<wsdl:message name="RunTestResponse">
<wsdl:part name="RunTestReturn" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="TestStatusResponse">
<wsdl:part name="TestStatusReturn" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="GetSnapshotResponse">
<wsdl:part name="GetSnapshotReturn" type="apachesoap:Map"/>
</wsdl:message>

<wsdl:message name="PauseTestResponse">
<wsdl:part name="PauseTestReturn" type="xsd:boolean"/>
</wsdl:message>

<wsdl:message name="ResumeTestResponse">
<wsdl:part name="ResumeTestReturn" type="xsd:boolean"/>
</wsdl:message>

<wsdl:message name="StopTestResponse">
<wsdl:part name="StopTestReturn" type="xsd:boolean"/>
</wsdl:message>
```

Develop a Class to Connect to PAAS

This is an interface between the application and PAAS. It is available as a standalone library for clients to connect their applications to PAAS Farms. Class Method describes the class methods:

Class Method

Class	Description
PAASClient(String ServerURL, String Username, String Password)	The class constructor; creates a new object for connecting to PAAS. The username and password are stored for future calls. The ServerURL is usually: ***http//ServerName:8080/TestNetwork/ws/PAAS***
public int doRunTest (String[]filenames, StringsScript)	Calls PAAS RunTest and returns the test status. Requires an array of files required by the scenario and the filename to be executed (only the name, not the full path). Generates a java.lang.Exception if the Scenario fails.
public int doTestStatus ()	Calls PAAS TestStatus to get the test status.
public boolean doPauseTest ()	Calls PAAS PauseTest and returns the test status.
public boolean doResumeTest ()	Calls PAAS ResumeTest and returns the test status.
public boolean doStopTest ()	Calls PAAS StopTest and returns the test status.

Example implementation

This code snippet shows a simple communication connection between an application and PAAS:

```
PAASClient client= null; // First we create the client
String [] files = new String[2]; // We add the required files
files[0] = "./SMSMonitor.xml"; // The Scenario
files[1]= "SendSMS.jar"; // The resources needed
String GSRSFile = "GSRS.xml"; // The file to save the GSRS

try

{
/* We create a client using the localhost as the server
* with Appvance as the username and paas as the password
*/
client = new PAASClient(
"http://localhost:8080/TestNetwork/ws/PAAS",
"Appvance",
"paas");

// We launch the test using the SMSMonitor.xml as the
// running scenario
client.doRunTest(files,"SMSMonitor.xml");

// If nothing goes wrong we inform of the success
System.out.println("Connection to PAAS Done");
}
catch (Exception e)
{
// Something goes wrong, we inform the situation and quit
e.printStackTrace();
System.out.println("Connection to PAAS Failed");
System.exit(0);
}

/*
* Every amount of time we read the current GSRS and show it in
* the screen
*/
try
{
while (true)
{
// Get the Governance file
client.doGetSnapshot(GSRSFile);
BufferedReader buff = new BufferedReader(new java.io.FileReader(src));
String line = buff.readLine();
while (line != null)
//Prints lines of the of the Governance
System.out.println(line);
try { Thread.sleep(10000); } catch (Exception e){}
}
}
catch (Exception e)
{
//Errors obtaining the GSRS, we show it and quit
System.out.println("Error adding the GSRS");
System.exit(0);
```

We found these resources helpful while implementing PAAS:

Microsoft Developer Network: Software as a service

Microsoft Developer Network: Architecture Strategies for Catching the Long Tail

Anderson, Chris: The Long Tail

PAAS Response

Each PAAS request to Appvance IQ generates a PAAS response:

PAAS Request / Appvance IQ Response

PAAS Request	Appvance IQ Response
RunTest	Returns the session ID number. Request body is a TestScenario XML document.
TestStatus	Requires session ID value from the RunTest operation. Returns a single value indicating the test status: 0 - Test operating normally. 1 - Test running and encountered one or more exceptional conditions. 2 - Test paused. 3 - Test ended. 4 - Problems (test does not exist, authentication failed).
GetSnapshot	Requires session ID from the Runtest operation. Returns a GSRS formatted XML document containing the test's most recent result.
PauseTest	Requires session ID from the RunTest operation. Puts test in "pause" condition.
ResumeTest	Requires session ID from the RunTest operation. Changes test condition from "pause" to"run".
StopTest	Requires session ID from the RunTest operation. Puts test in "stop" condition.

The corresponding Web service result values are:

Web Service Result Values

Function Name	Response Value
RunTest	0 - Error; something went wrong. >0 - Session ID for the scenario.
TestStatus	Test status value.
GetSnapshot	GSRS formatted XML. NULL - Error (no such id. authentication failed).
PauseTest	True - Test has been paused. False - Unable to stop the test (no such id. authentication failed).
ResumeTest	Trus - Test has been resumed. False - Unable to resume test (no such id. authentication failed).
StopTest	True - Test has been stopped. False - Unable to stop the test (no such id. authentication failed).

Governance Statistics ResultSet (GSRS)

Appvance executes a test and returns a Governance Statistics Results Set (GSRS) response document.

A sample GSRS response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<service>
<status>
<lastchecked>Fri Jun 22 10:51:51 CST 2007</lastchecked>
<state>up</state>
</status>

<statistics>
<timesincelastfailure measurement="minutes">17</timesincelastfailure>
<responsetime measurement="milliseconds">
<average>30082</average>
<minimum>0</minimum>
<maximum>30673</maximum>
<testcount>26</testcount>
</responsetime>
<reliability/>
<function>
<comments/>
</function>
</statistics>

<accepts type="XML schema">
<wsdl>http://examples.pushtotest.com/axis/services/MessageService?wsdl</wsdl>
<wsdl>http://examples.pushtotest.com/axis/services/MessageService?wsdl</wsdl>
<wsdl>http://examples.pushtotest.com/axis/services/MessageService?wsdl</wsdl>
</accepts>

<security type="Public Certificate">
<certificate>38ADKJF83JF838DJQIEUR3</certificate>
<id>fcohen</id>
<note>Accepts PGP and OpenSSL certificates</note>
</security>
</service>
```

If you have any feedback on this topic, please contact us at documentation@appvance.ai

Last updated: October 15, 2026

Revision #3
Created 2026-07-31 16:29:44 UTC by Admin
Updated 2026-10-15 20:53:25 UTC by mmora