PowerMatcher Core

Test Report

Authors:

Carlo Haverkamp (IBM)

Release: 0.9

Revision: 2

Date: 14-2-2013

Table of Contents

Version History

Revision	Description	Author
0.9 - 1	Initial version	C. Haverkamp (IBM)
0.9 - 2	Added further clarification to section 3.1 IAF3	A.H. Eisma (IBM)

1 Introduction

1.1 Identification

The full identification of this document is: “PowerMatcher Core - Test Report”.

1.2 Background

This document describes the results of the scenarios that have been written to test the PowerMatcher code (see [1] and [2])

1.3 Purpose and scope

This document contains the a short summary of the performed tests and the results. It provides also information on the test implementations and where they can be found.

1.4 Intended Audience

Managers, architects, developers and testers.

2 Test approach

The test method and test design of the tests described in this document are based on the *PowerMatcher Test Plan*[1].

The focus in the implementation of these tests was to automate the tests using scripts to support future regression tests. All Intelligence tests and some Communication Framework tests have been implemented using JUnit.

The JUnit tests read test input and expected results from csv-files. When running the tests within Eclipse the test results can be verified in the JUnit view.

The scenarios that test adding, updating, deleting agents, concentrators and auctioneers need to be run within a cluster. For these tests OSGi JUnit classes have been implemented. In these tests an OSGi runtime is started and stopped for the duration of a test. Triggering events like updating agents is realized by loading configuration files into the ConfigManager which will update the OSGi configurations.

An OSGi JUnit test script for PowerMatcher uses the following approach:

- Configuration data in XML is read and processed by the ConfigManager. By using multiple configuration XML files configuration updates can be simulated.
- Automated checking of configuration updates.
- Automated state checking of components (e.g. running agents)
- Automated checks of log file information.

2.1 Starting the tests

The JUnit tests can be started from a test suite class named ‘MainTestSuite’ in the net.powermathcher.core.test project.

All OSGi JUnit tests can be run from the bnd run configuration file ‘PwM Core OSGi Junit Run.bndrun’ (Run as -> OSGi JUnit test) in the project net.powermatcher.core.test.osgi.

3 Intelligence testing

This chapter provides an overview of the Intelligence tests and the results.

3.1 Auctioneer scenarios

The table below lists the Functional and Quality tests for the Auctioneer component. The tests have been implemented in JUnit java classes. The following classes in the project net.powermatcher.core.test implement the Auctioneer scenarios:

- net.powermatcher.core.agent.auctioneer.test.AuctioneerResilienceAFPriceTests.java
- net.powermatcher.core.agent.auctioneer.test.AuctioneerResilienceTestAF.java
- net.powermatcher.core.agent.auctioneer.test.AuctioneerResilienceTestIAQ1.java
- net.powermatcher.core.agent.auctioneer.test.AuctioneerResilienceTestIAQ2.java
- net.powermatcher.core.agent.auctioneer.test.test/AuctioneerTestSuite.java

Running test suite class will run all Auctioneer tests.

Validation method	Test ID	Scenario	Test result	Notes
Functional	IAF1	No equilibrium (demand side)	Success	
	IAF2	No equilibrium (supply side)	Success	
	IAF3	Equilibrium	Success	20-12-2012 Aggregated bid is ok. Test1 ok. Test2 failed: Equilibrium price is 41 but expected was 40. Should be 40 because at 40 demand is 907, while at 41 demand is -2863. Then price 40 results in the closest price to a balance. 08-02-2013 Test successful. Originally expected result in test scenario does not conform to the current protocol specification of price calculation. In the current specification and implementation the test data is interpreted as a vertical price step at 41, so the equilibrium price is 41. If the input to the test would be a such a step bid curve in point representation, it results in the demand array that is used in this test after conversion according to the current protocol specification. The expected outcome of the test scenario is updated. Original input file appended with .err extension in SVN.
	IAF4	Multiple consecutive equilibriums	Success	20-12-2012 Aggregated bid is ok. First sequence failed: Equilibrium price is 41 but expected was 40. Should be 40 because at 40 demand is 907, while at 41 demand is -2863. Then price 40 results in the closest price to a balance. 08-02-2013 Test successful. See explation in IA3.
	IAF5	Equilibrium including bid rejection	Success	20-12-2012 Ascending bids rejected. Calculated equilibrium was 6.0 but expected was 5.0. Should be 5.0 when demand is 535 while at 6.0 the demand is -2776. 535 closest to a balance. 08-02-2013 Test successful. See explation in IA3.
	IAF6 ... IAF12	Additional equilibrium tests	Success	20-12-2012 Tests IAF06-IAF09 ok. Tests IAF10-IAF12 failed. Failure reason is that the current implemented algorithm takes the highest price step when the equilibrium is between two price steps. Expected price is however the price that has a demand closest to 0 (smallest imbalance). 08-02-2013 Test successful. See explation in IA3.
Quality	IAQ1	IAF5	Success	
	IAQ2	Fault tolerance	Success	

The tests covering the equilibrium calculation (IAF3,…, IAF12) showed initially a difference in the expected price. However, it appeared that the implementation followed the current design specifications. The underlying cause is that representation of the demand can be described in point notation or in an array. Both notations should represent the same demand curve and conversion between the two should not lead to any differences. The test input for the scenario's used the array representation.

3.2 Concentrator scenarios

The table below shows the Functional and Quality tests for the Concentrator component. The tests have been implemented in JUnit java classes. The following classes in the project net.powermatcher.core.test implement the Concentrator scenarios:

- net.powermatcher.core.agent.concentrator.test.ConcentratorResilienceTestICF.java
- net.powermatcher.core.agent.concentrator.test.ConcentratorResilienceTestICQ.java
- net.powermatcher.core.agent.concentrator.test.ConcentratorTestSuite.java

Running test suite class will run all Concentrator tests.

All tests have run successfully.

Validation method	**Test ID**	**Scenario**	**Test result**	**Notes**
Functional	ICF1	Bid aggregation	Success	
	ICF2	Bid aggregation including bid rejection	Success	10-12-2012 Ascending bid not rejected. 21-12-2012 Ascending bids are rejected. Aggregation ok.
	ICF3	Price forwarding	Success	
Quality	ICQ1	Scalability	Success	Test with > 10000 bids from individual agents. Aggregation ok.
	ICQ2	Fault tolerance	Success	Test with > 10000 bids from individual agents. Aggregation ok.

4 Communication Framework Testing

This chapter provides an overview of the Commucation Framework tests and the results.

4.1 Sending and Receiving Price Scenarios

The table below describes the Sending and Receiving Price scenarios. The scenarios have been implemented in JUnit java classes. Test CPF2 has not been implemented. The others have run successfully.

The following classes in project net.powermatcher.core.test implement the tests:

- net.powermatcher.core.SendReceivePriceTestCPF1.java
- net.powermatcher.core.SendReceivePriceTestCPQ1.java

Validation method	Test ID	Scenario	Test result	Notes
Functional	CPF1	Send a price to an downstream component	Success	
	CPF2	Sending a price downstream after a marketbasis update		To be implemented.
Quality	CPQ1	Fault-tolerance	Success	18-12-2012 Auctioneer does not publish null price, but null price is stored as last published price. 18-12-2012 Acutioneer publishes price (52) outside price range [0,50] 18-12-2012 Concentrator does not reject price (52) outside price range [0,50] 18-12-2012 Concentrator forwards price (52) outside price range [0,50] to agents. 18-12-2012 Agents do not reject price (52) outside price range [0,50] 08-02-2013 Test successful. Discussed results with Aldo Eisma. The auctioneer is allowed to send prices outside the price range because it is possible that at the destination the ere can be a market base with a different price range.

4.2 Sending and Receiving Bid Scenarios

The table below describes the Sending and Receiving Bid scenarios. The scenarios have been implemented in JUnit java classes. Test CBF2 has not been implemented. The others have run successfully.

The following classes in project net.powermatcher.core.test implement the tests:

- net.powermatcher.core.SendReceiveBidTestCBF1.java
- net.powermatcher.core.SendReceiveBidTestCBQ1.java

Validation method	Test ID	Scenario	Test result	Notes
Functional	CBF1	Send a bid to an upstream component	Success	
	CBF2	Sending a bid upstream after a marketbasis update	-	Not implemented
Quality	CBQ1	Fault-tolerance	Success	Difficult to send invalid bids in an agent configuration since BidInfo prevents creation of invalid bids.

4.3 Adding, Updating or Removing Auctioneer Scenarios

These scenarios cover testing of the Auctioneer running in a cluster. The tests have been implemented in OSGi JUnit classes and can be run as ‘OSGi JUnit test’ which starts an OSGi environment, runs the test and finally shuts down the OSGi runtime.

The following classes in project net.powermatcher.core.test.osgi implement the tests:

- net.powermatcher.core.test.osgi.ResilienceTestCAF.java
- net.powermatcher.core.test.osgi.ResilienceTestCAQ.java

All tests within the project can be started from the bnd run configuration file ‘PwM Core OSGi Junit Run.bndrun’.

Two quality tests (CAQ1, CAQ2) have failed. Both have been confirmed as a problem an need to be fixed.

Validation method	**Test ID**	**Scenario**	**Test result**	**Notes**
Functional	CAF1	Add auctioneer (no other elements present)	Success	Auctioneer is added and is activated successfully. Direct agent protocol adapters are used in test. TODO: Repeat test using MQTT adapters.
	CAF2	Add auctioneer (agents and / or concentrators present)	Success	Auctioneer is added success-fully. All agents in cluster are operational. Direct agent protocol adapters are used in test..
	CAF3	Update auctioneer	Success	Auctioneer configuration and Auctioneer component prop-erties are updated success-fully. Cluster works fine.
	CAF4	Remove auctioneer	Success	Auctioneer configuration is removed and Auctioneer component is stopped ('un-satisfied'). Other components remain active. Concentrator stops pubishing bids.
Quality	CAQ1	Add an auctioneer to a cluster with an auctioneer (with or without the same identifier) already present.	Failed	Second auctioneer is suc-cessfully added and is being activated, which is not the expected outcome. Cluster remains working, but unclear if this impacts correct func-tioning. Direct agent protocol adapters are used in test. Analysis: OSGi permits this, therefore the ConfigManager or another mechanism should be added to prevent this.
	CAQ2	Update an auctioneer with bad configuration data (i.e. configuration data containing parameters which are not inside allowed limits).	Failed	The properties of the Auc-tioneer component contain the invalid values, so this seems not ok. However, the auctioneer is still active. Di-rect agent protocol adapters are used in test. Analysis: Property updates are updated. For example in case of an update.interval=-30 this would lead to an ex-ception when binding to the scheduler adapter. The component is active but doing nothing.

	CAQ3	Ungracefully remove an auctioneer from a cluster.	Success	The auctioneer is removed and from the logs we can see that no bids are sent by the concentrator, though the TestAgent continues to send bids. Other agents remain active. Direct agent protocol adapters are used in test.
	CAQ4	Ungracefully remove an auctioneer from a cluster, and after a set time, re-add an auctioneer to the cluster.	Success	The auctioneer is removed and from the logs we can see that no bids are sent by the concentrator, though the TestAgent continues to send bids. Other agents remain active. Whe the auctioneer configuration is restored price info is sent again and con-centrator sends bids to auc-tioneer. Cluster is then work-ing fine. Direct agent protocol adapters are used in test.

4.4 Adding, Updating or Removing Concentrator Scenarios

These scenarios cover testing of a Concentrator running in a cluster. The tests have been implemented in OSGi JUnit classes and can be run as ‘OSGi JUnit test’ which starts an OSGi environment, runs the test and finally shuts down the OSGi runtime.

The following classes in project net.powermatcher.core.test.osgi implement the Concentrator tests:

- net.powermatcher.core.test.osgi.ResilienceTestCCF.java
- net.powermatcher.core.test.osgi.ResilienceTestCCQ.java

All tests within the project can be started from the bnd run configuration file ‘PwM Core OSGi Junit Run.bndrun’.

Two quality tests (CCQ1, CCQ2) have failed. Both have been confirmed as a problem an need to be fixed.

Valida-tion method	**Test ID**	**Scenario**	**Test result**	**Notes**
Functional	CCF1	Add concentrator (no other elements present)	Success	Concentrator is added and is activated successfully. Direct agent protocol adapters are used in test.
	CCF2	Add **concentrator** (agents, auctioneers and / or con-centrators present)	Success	Concentrator is added successfully. Communication startes when concentra-tor is added. Direct agent protocol adapters are used in test..
	CCF3	Update concentrator	Success	Concentrator configuration and Concen-trator component properties are updated successfully. Cluster works fine.
	CCF4	Remove concentrator	Success	Concentrator configuration is removed and Concentrator component is stopped ('unsatisfied'). Other components remain active but no bid/price communication. .
Quality	CCQ1	Add an concentrator to a cluster with a concentrator with the same identifier already present.	Failed	Second concentrator with same id is successfully added and is being acti-vated, which is not the expected out-come. Cluster remains working, but unclear if this impacts correct function-ing. Direct agent protocol adapters are used in test. Analysis: OSGi permits this, therefore the ConfigManager should be modified to prevent this.

	CCQ2	Update a concentrator with bad configuration data (i.e. configuration data containing parameters which are not inside allowed limits).	Failed	Concentrator configuration is updated with the invalid values and the concentrator component is also updated with the invalid property values and remains Active. Analysis: Property updates are updated. For example in case of an update.interval=-30 this would lead to an exception when binding to the scheduler adapter. The component is active but doing nothing.
	CCQ3	Ungracefully remove a concentrator from a cluster.	Success	The concentrator is removed and from the logs we can see that no bids are sent by the test agent. The auctioneer remains active and logs the last aggregated bid. Direct agent protocol adapters are used in test.
	CCQ4	Ungracefully remove a concentrator from a cluster, and after a set time, re-add a concentrator to the cluster.	Success	The concentrator is removed and from the logs we can see that no bids are sent by the test agent. The auctioneer remains active and logs the last aggregated bid. The third configuration update restores the concentrator and the cluster is again fully operational. Direct agent protocol adapters are used in test.
	CCQ5	Ungracefully remove an concentrator from the cluster after it has sent a bid, but before the upstream component has calculated either a price (auctioneer) or aggregated bid (concentrator).	-	To be implemented.

4.5 Adding, Updating or Removing Agent Scenarios

These scenarios cover testing of a TestAgent running in a cluster. The tests have been implemented in OSGi JUnit classes and can be run as ‘OSGi JUnit test’ which starts an OSGi environment, runs the test and finally shuts down the OSGi runtime.

The following classes in project net.powermatcher.core.test.osgi implement the TestAgent tests:

- net.powermatcher.core.test.osgi.ResilienceTestCGF.java
- net.powermatcher.core.test.osgi.ResilienceTestCGQ.java

All tests within the project can be started from the bnd run configuration file ‘PwM Core OSGi Junit Run.bndrun’.

Two quality tests (CGQ1, CCG2) have failed. Both have been confirmed as a problem an need to be fixed.

Validation method	**Test ID**	**Scenario**	**Test result**	**Notes**
Functional	CGF1	Add agent (no other elements present)	Success	TestAgent is added and is activated successfully. Direct agent protocol adapters are used in test.
	CGF2	Add agent (agents, auctioneers and / or concentrators present)	Success	TestAgent is added successfully. Communication starts when test agent is added. Direct agent protocol adapters are used in test..
	CGF3	Update agent	Success	TestAgent configuration and TestAgent component properties are updated successfully. Cluster works fine.
	CGF4	Remove agent	Success	TestAgent configuration is removed and TestAgent component is stopped ('unsatisfied'). Other components remain active but no bid/price communication. .

Quality	CGQ1	Stability	Failed	Second test agent with same id is successfully added and is being activated, which should not be possible. Cluster remains working, but unclear if this impacts correct functioning. Direct agent protocol adapters are used in test. Analysis: OSGi permits this, therefore the ConfigManager should be modified to prevent this.
	CGQ2	Stability	Failed	TestAgent configuration is updated with the invalid values and the test agent component is also updated with the invalid property values and remains Active. The bind of the scheduler fails due to an IllegalArgumentException.
	CGQ3	Stability	Success	The test agent is removed and from the logs we can see that no bids received by the concentrator. The auctioneer and concentrator remain active and log the last aggregated bid. Direct agent protocol adapters are used in test.
	CGQ4	Recoverability	Success	The test agent is removed and from the logs we can see that no bids received by the concentrator. The auctioneer and concentrator remain active and log the last aggregated bid. Direct agent protocol adapters are used in test. The third configuration update restores the test agent and the cluster is again fully operational. Direct agent protocol adapters are used in test.
	CGQ5	Stability		To be implemented.

4.6 General Quality Scenarios: Fault-tolerance, Scalability, Stability and Recoverability

The General Quality Scenarios have not been implemented.

Validation method	**Test ID**	**Scenario**	**Test result**	**Notes**
Quality	CQ1	Send bad / corrupted prices and bids - both small and huge amounts - throughout a cluster.		To be implemented.
	CQ2	Send garbage data - both small and huge amounts - throughout a cluster.		To be implemented.
	CQ3	Run all price and bid sending scenarios for a huge amount of prices and bids in a short amount of time.		To be implemented.
	CQ4	Run all addition, update and removal scenarios for a huge amount of components in a short amount of time.		To be implemented.
	CQ5	Remove key communication components (e.g. message bus components).		To be implemented.
	CQ6	Block either bid or price sending / receiving paths of a single cluster element, e.g. prices can be received but bids cannot be sent		To be implemented.
	CQ7	Remove key communication components (e.g. message bus components), and re-add them after a certain time.		To be implemented.

5 Unit tests

In the development phase of PowerMatcher there have also unit tests have been written. These unit tests have also been included in a test suite class *PwmCoreUnitTestSuite* in project net.powermatcher.core.test. This script is also invoked from the MainTestSuite JUnit test suite script in the same project.

The following test scripts are started from PwmCoreUnitTestSuite:

- net.powermatcher.core.data.test.BidInfoTest;
- net.powermatcher.core.data.test.PriceInfoTest;
- net.powermatcher.core.messaging.framework.test.TopicTest;
- net.powermatcher.core.messaging.protocol.adapter.test.han.test.HANBidMessageTest;
- net.powermatcher.core.messaging.protocol.adapter.test.han.test.HANPriceInfoMessageTest;
- net.powermatcher.core.messaging.protocol.adapter.test.internal.test.InternalBidMessageTest;
- net.powermatcher.core.messaging.protocol.adapter.test.internal.test.InternalPriceInfoMessageTest;
- net.powermatcher.core.messaging.protocol.adapter.test.log.test.BidLogMessageTest;
- net.powermatcher.core.messaging.protocol.adapter.test.log.test.PriceLogMessageTest;

6 Recommendation for future testing

The tests in the PwM Resilience Testing.xls planning sheet [2] have not all been implemented and executed. We recommend that in another phase:

- Implemented and perform the remaining tests.
- Implement alternative scenarios for the configuration tests using indirect adapters (include the MQTT message bus infrastructure in the tests).
- Regression tests after problem fixing or new development.

7 References

[1]	PowerMatcher Test Plan, Version 0.5; Mark Bastiaans (TNO), Pamela MacDougall (TNO), Carlo Haverkamp (IBM)	01/11/12
[2]	Resilience Testing planning and testing excel sheet, PwM Resilience Testing.xls; Mark Bastiaans (TNO), Pamela MacDougall (TNO), Carlo Haverkamp (IBM)	07/02/2013