0. Cover

ORIGINATOR: TAXUD/R5	ISSUE DATE: 01.06.2012	VERSION: 1.00

ANNEX II.D - CCN2 PLATFORM PROOF OF CONCEPT SPECIFICATIONS

Invitation to tender TAXUD/2011/AO-013

Specification, development, maintenance and 3rd level support of CCN and CCN2

(CCN2-DEV)

Typographic conventions

The following typographic conventions are used in this document:

⚠	Draws attention to important information
ℹ	Indicates definitions or reference information
📝	Indicates that this requirements must be clearly addressed in the tender

Table of Contents

0.1. Table of Tables

0.2. Table of Figures

0.3. Introduction

0.3.1. *Purpose*

This document is Annex II.D of this Call for Tenders and provides the **CCN2-DEV** CCN2 Platform Proof of Concept Specifications.

0.3.2. *Overview*

This Annex II.D of this Call for Tenders has the following structure:

SECTION	DESCRIPTION
Chapter 0	This chapter
Chapter 1	Proof of Concept – Overview
Chapter 2	Proof of Concept Specification.
Chapter 3	Test Scenarios

Table 1: Document structure

0.4. Acronyms and definitions

An extensive list of abbreviations can be found in "Annex II.E - List of Abbreviations and Definitions".

1. Proof of Concept - Overview

As part of the award criteria for the CCN2 Platform, Tenderers are requested to develop a Proof of Concept for their proposed solution.

The Proof of Concept must be developed using the guidelines which are outlined here.

DG TAXUD will evaluate and score the Proof of Concept as a key part of the award criteria and therefore it is essential that:

- Clear documentation is provided in terms of the design of the Proof of Concept and the resulting reports;
- Full transparency is provided so that DG TAXUD can clearly see and validate the operation of the Proof of Concept;
- The Proof of Concept is fully testable by DG TAXUD using both positive and negative test scenarios which have been developed and run by the Tenderer (i.e. DG TAXUD reserves the right to re-run some or all of the tests as indicated in the Tendering Specifications linked to this Call for Tenderers, section 6.3.1).

1.1. Objectives

The objective of the Proof of Concept is to demonstrate key aspects of the CCN2 Platform with respect to:

- Functionality of the key modules;
- Integration between the key modules;
- Resilience;
- Performance;
- End to end security;

End to end operations must be demonstrated and therefore the design of the Proof of Concept must include:

- Applications that provide services that are made available on the CCN2 Platform;
- Applications that consume services which are made available on the CCN2 Platform.

1.2. Scenario Overview

The goal of the Proof of Concept is to demonstrate as much of the functionality of the underlying components which make up the CCN2 Platform as possible. Given time limitations it is understood that not all functionality can be demonstrated but the scope must include the scenarios outlined in the table below (the wording for the classification is described below):

SCENARIO	TITLE	CLASSIFICATION
1	Deployment of a Web Service on the SOA Backplane	**MUST**
2	Use of the Web Services in a simulated environment	**MUST**
3	Logging and traceability of a Web Service	**MUST**
4	Messages' Persistence and Recoverability	**MUST**
5	Dynamic monitoring of a Web Service	**SHOULD**
6	SOA Governance Policy Set up and Run Time Enforcement	**SHOULD**
7	Performance and Scalability	**SHOULD**
8	IAM Module integration	**MUST**
9	Access Point Failover	**MUST**
10	Platform Failover	**MUST**
11	Master Data Management	**SHOULD**

Table 2: Definition of Access Point

The following wording is used to denote the importance of requirements:

CLASSIFICATION	DESCRIPTION
MUST	It is mandatory for a solution proposed by the Tenderer to comply with the requirement, feature or behaviour.
SHOULD	The requirement is recommended but it is not mandatory. However, it is a Tenderer's decision whether to demonstrate a solution depending on various factors such as cost, complexity, reliability, availability of the features in the products, and amount of custom development involved. As part of the questionnaire the Tenderer will be asked to justify the decision taken.
MAY	Not applicable in this document.

Table 3: Wording for the requirements

1.3. Conceptual Architecture

The conceptual architecture for the Proof of Concept is given in the diagram below.

For the Proof of Concept the Tenderer will have to develop a number of small applications which sit outside the CCN2 Platform:

- Application A (App.A) is a Web Service Consumer;
- Application B (App.B) is a Web Service Provider;
- Application C (App.C) is an Event Sink.

Details for the specifications of the applications are presented further below.

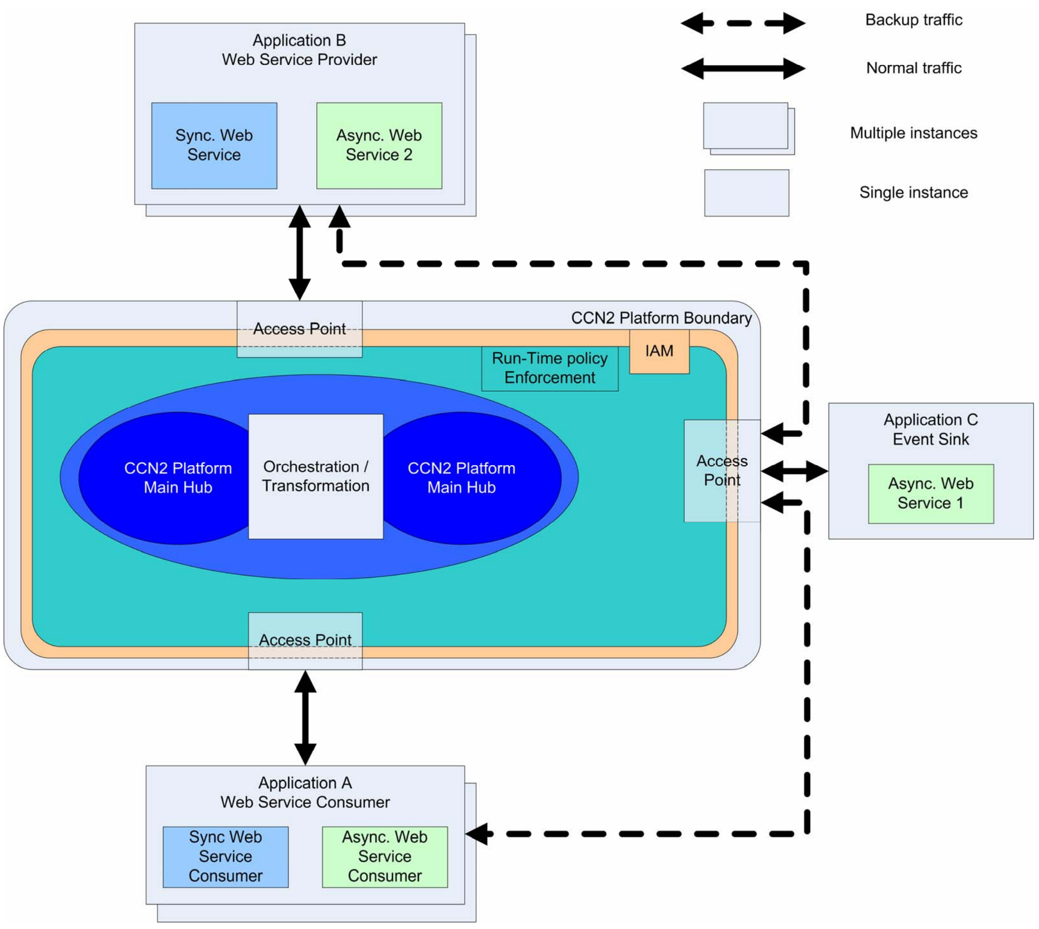

Figure 1: Interoperability Platform

To demonstrate the MDM capabilities of the CCN2 Platform a federated data set must be set up where Organisation A manages certain elements of the master data and Organisation B manages other elements of master data. The master data must be synchronised for any changes made by Organisation A or B and any data changes must be replicated to Organisation C who has no ability to change master data elements but must receive copies of all changes.

The Tenderer is free to define the data set used for this section.

1.4. Documentation

It is expected that the Tenderers will provide sufficient documentation covering all activities in this PoC, which includes but is not limited to:

1. Proof of Concept architecture design;
2. Software and hardware bill of materials;

> ⚠ The use of the Infrastructure Description Form (IDF) (see Annex I.A – Questionnaire, attachment 3 – Infrastructure Description Form) is mandatory in this context of the ICT description in the bill of materials.

3. Technical specifications for the Web Services in terms of how they are designed and coded (the internal design of the applications is not of interest);
4. Platform customisation and configuration description (e.g. including transformation maps if were used);
5. Master Test Plan: Test Strategy and Approach that describes the approach, scope, activities, roles and responsibilities in order to successfully test the PoC;
6. Test Design Specifications: Description of the Test scenarios including identifier, purpose, prerequisites, inputs specification, test steps, expected results and pass fail criteria (one document describing all Test scenarios);
7. Test Reports (one document describing all Test reports).

2. Proof of Concept Specifications

2.1. Setup the Proof of Concept (PoC)

The Tenderer must design, setup, configure the PoC environment, and develop any required components and customisations in order to provide the needed hardware and software environment for the test scenarios to be executed.

The methodologies and tools (development, governance, testing, etc.) that are proposed for the development of the CCN2 Platform must be used for the PoC activities (including the development stub applications and Web Services). Any deviations must be clearly stated and explained.

The architecture for the Proof of Concept may be scaled down from the future CCN2 Platform production configuration at the Tenderers discretion but it must only contain components which have been proposed for the CCN2 Platform.

The PoC may be used to also demonstrate the separation between development and production environments.

The setup of the PoC also requires the creation of all needed users and roles used by the test scenarios (including the MDM). The processes used are documented in the appropriate documents.

This must be documented as described above.

2.2. Specifications of the applications

The specifications of the applications that need to be developed are as follows:

1. A stub application (App.B) which provides a Web Service. The implementation of this Web Service must be via a pre SOA adapter linked to an API supported by the application (in other words a service will be exposed by the CCN2 Platform encapsulating API calls via an adapter module). The stub application will support:
    a. A single synchronous (request/response) Web Service. However the CCN2 Platform will support multiple versions of this Web Service which have differences in their characteristics. In other words the CCN2 Platform is able to transform the output of App.B thereby providing different responses depending on which version of the Web Service is called and;
    b. A single asynchronous Web Service which passes a message to App.B. App.B will generate a response but App.A will not wait for the response which can be generated and set later. However the CCN2 Platform will be responsible for generating a delivery message once App.B has accepted the Web Services call.
2. A stub application (App.A) which acts as service consumer to call the previously defined Web Service with the appropriate authentication. App.A must be capable of calling multiple versions of synchronous Web Services which are to be developed as well as the asynchronous Web Service (refer to section 2.3 of this document).
3. A stub application (App.C) which acts as an event sink where messages can be picked up for further processing (note that further processing of the information is not within

the scope of the Proof of Concept). Messages will be delivered to the stub application via an asynchronous Web Service on the successful completion of the synchronous Web Service calls.

Please note the following points:

- Application A (App.A) must be capable of being deployed as multiple instances (i.e. capable of multiple simultaneous Web Service calls to App.B).
- Application B (App.B) must be capable of being deployed as multiple instances (i.e. capable of receiving multiple simultaneous Web Service calls from App.A).
- Application C (App.C) is intended as an event sink node and may process messages sequentially.

2.3. Introduction to flows

The purpose if this section is to present graphically the two main flows needed in the Web Services exchanges (the Web Services will be specified in the next section):

- The synchronous flow between App.A and App.B (with the App.C. as event sink);
- The asynchronous flow between App.A and App.B.

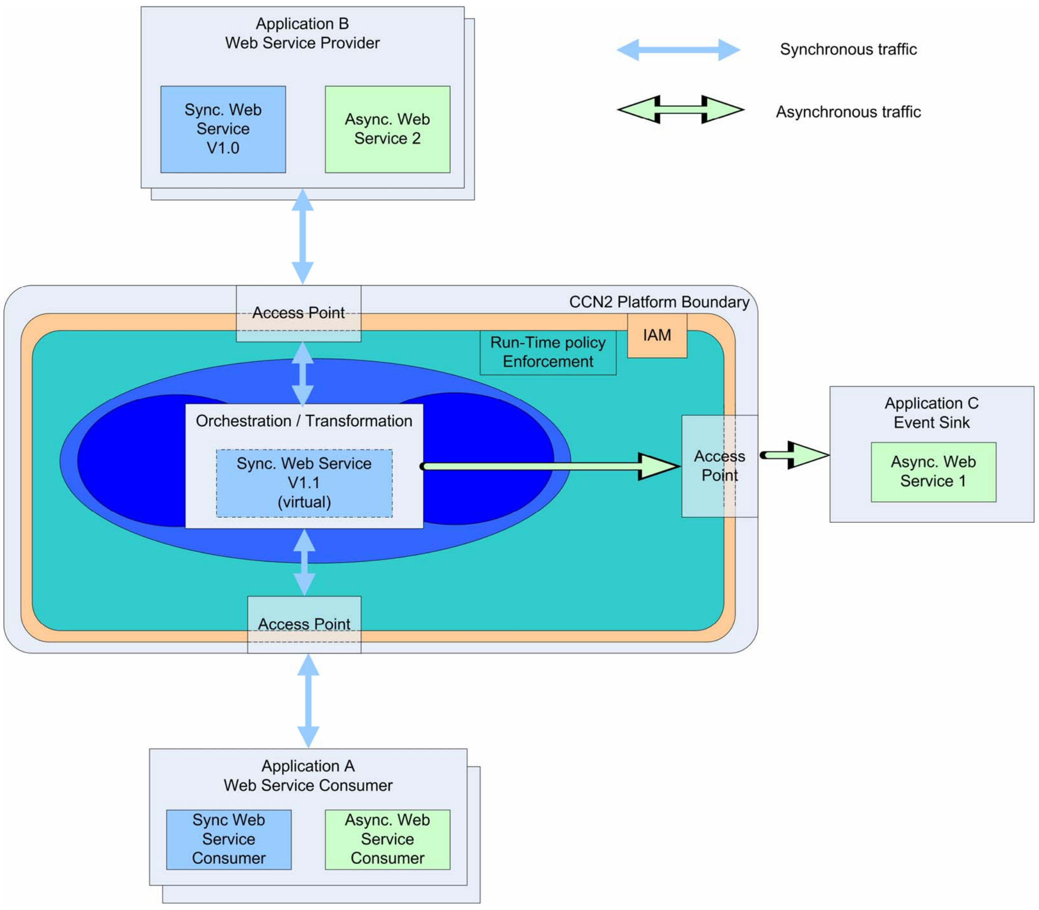

Figure 2: Synchronous flow with orchestration/transformation

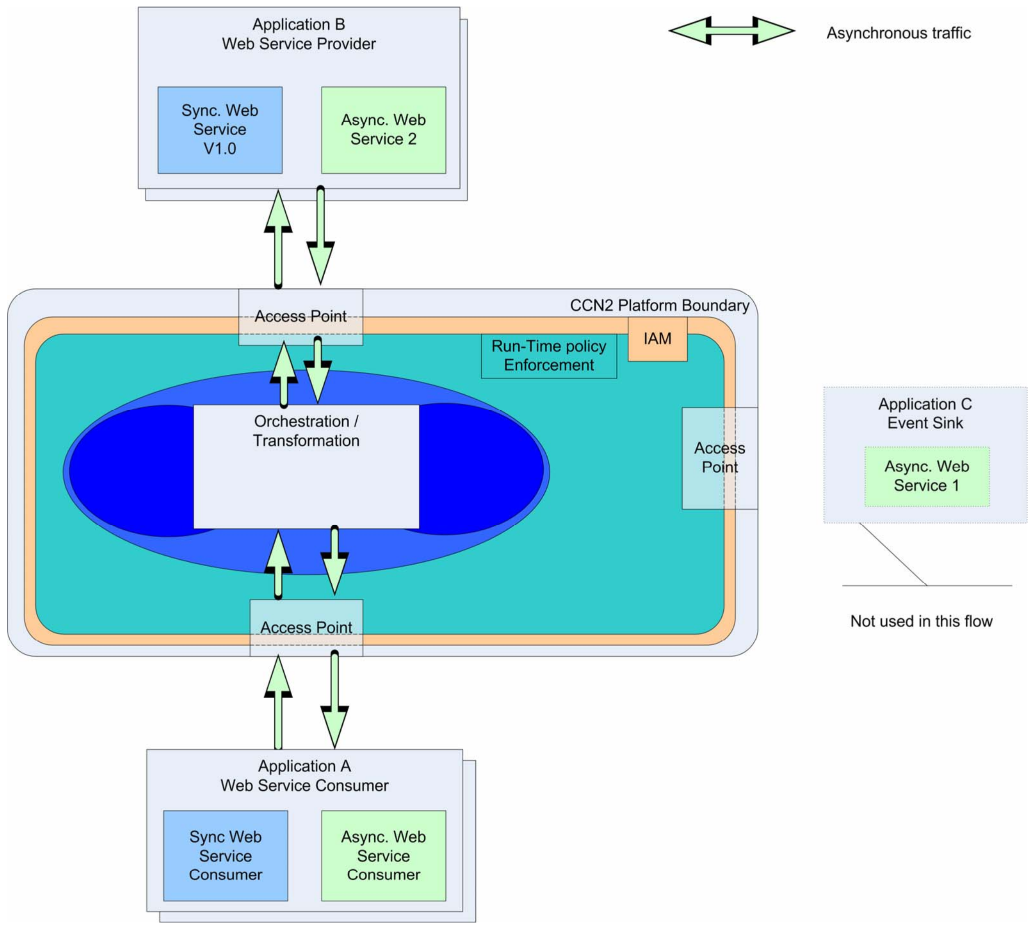

Figure 3: Asynchronous flow with transformation

2.4. Web Services Specification

2.4.1. *App B's Synchronous Web Service Version 1.0*

Synchronous Web Service version 1.0 must be designed in a request/response manner whereby a sender sends a request to a receiver who invokes an external business application. The business application processes the request and generates a response, which is returned to the sender that originated the request.

The following are the key characteristics that must be developed as part of the synchronous Web Services version 1.0:

1. Conforms with the following WS specifications:
   a. SOAP
   b. UDDI
   c. WSDL
   d. WS-Security
2. XML Message based payload which is validated against an XML schema and rejected if not correct (no message repair);

3. Encryption of the message payload;
4. Authentication of the user credentials (consumer) to use the service against the IAM Module;
5. Transformation of message content based on defined transformation rules;
6. Non-repudiation of the service usage and message delivery;
7. The business application invoked by the service must have its functionality exposed via an adapter developed with the CCN2 Platform adapter kit. This adapter must map the Web Services call to a pre SOA application via a defined API (N.B. it is not necessary to use the CSI API in the Proof of Concept. Any pre SOA API may be used as the objective is to demonstrate the power of the Adapter development kit in exposing business services via pre SOA APIs).

2.4.2. *Synchronous Web Service Version 1.1*

Web Services version 1.1 must be identical to that of version 1.0 except for the following points:

1. Authentication credentials will not be required to call the Web Services;
2. Some confidential aspects of the data returned by the business application will be masked by the Web Service;
3. The business application will not change (it only provides one service), thus the SOA Backplane needs to mask the data that is to be returned to the requestor.

2.4.3. *Asynchronous Web Service 1*

The asynchronous Web Service 1 must be designed from an application's perspective in a way that allows the sender to send an unacknowledged message to a single receiver. However the receipt of the message by the receiver must trigger the CCN2 Platform to generate an internal event to acknowledge that the message has been delivered but no response is required on the part of the receiver.

The following are the key characteristics that must be developed as part of the asynchronous Web Service 1:

1. It is only called on successful completion of the synchronous Web Service version 1.0 or version 1.1. In other words App.A must make a synchronous Web Service call to App.B which is successfully completed before the message is send to App.C;
2. The orchestration of the asynchronous Web Service 1 must be managed by the SOA Backplane (i.e. the microflow which calls the asynchronous Web Services only on the successful completion of the synchronous Web Service);
3. The asynchronous Web Service 1 takes the response from the synchronous Web Services and carries out the following tasks:
   a. Transform the output according to a predefined transformation map;
   b. Mask some of the output which is deem confidential;
   c. Deliver the output to a queue;

4. App.C takes the output from the queue and write it to a database for persistent storage;
5. Once the output has been taken from the queue an event (internal to the CCN2 Platform) is generated for logging and tracing purposes.

2.4.4. *Asynchronous Web Service 2*

The following are the key characteristics that must be developed as part of the asynchronous Web Services 2:

1. The asynchronous Web Service 2 is hosted on App.B and is called by App.A;
2. Once the request has been accepted by the CCN2 Platform, events are generated for logging and tracing purposes;
3. The security of the asynchronous Web Service 2 is managed by the SOA Backplane;
4. Once the request has been accepted by the App.B, internal events are generated for logging and tracing purposes, regardless of the presence of an answer;
5. App.B handles the request and prepares a response. The delay to generate the response can be random and App.A will not wait for a response;
6. Once the response has been accepted by the CCN2 Platform, internal events are generated for logging and tracing purposes;
7. The orchestration layer of the SOA Backplane takes the response from the App.B and carries out the following tasks:
    a. Transform the output according to a predefined transformation map;
    b. Mask some of the output which is deemed confidential;
    c. Deliver the response to the receiver;
    d. Note: It must be clear that the same processing can be applied on the request (even if not demonstrated).
8. App.A handles the response and writes it to a persistent storage.
9. Once the response has been accepted by the App.A, internal events are generated for logging and tracing purposes.

3. Test Scenarios

The scenarios (excluding the scenario relating to Master Data Management) are interdependent in that the Web Services developed are used through all the scenarios.

3.1. Deployment of a Web Service on the SOA Backplane

3.1.1. *Objective*

The objective of this scenario is to demonstrate how the Web Services can be deployed in the production environment.

3.1.2. *Approach*

The previously developed Web Services will be deployed on the SOA Backplane. It must be clearly demonstrated that the deployment does not require direct access between the development and production environments but the result of the development is a package which can be installed on the production environment.

3.1.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;
3. Deployment Plan.

3.1.4. *Steps*

The following steps must be completed for this scenario:

1. Demonstrate the deployment of all the Web Services on the SOA Backplane;
2. Demonstrate how the repository/registry can be searched to find the Web Services interface and binding descriptions for all the different Web Services.

3.1.5. *Output*

Test Report.

3.2. Use of the Web Services in a simulated environment

3.2.1. *Objective*

The objective of this scenario is to demonstrate that the Web Services function in accordance with the defined functional requirements.

3.2.2. *Approach*

The approach taken must conform to standard user acceptance testing as defined by the Tenderer's software development methodology.

3.2.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;

3.2.4. *Steps*

1. The following must be tested:
   a. The calling of both synchronous Web Services from App.A;
   b. The response from the App B's synchronous Web Services;
   c. The calling of the App C's asynchronous Web Service 1 on completion of a synchronous Web Service call;
   d. The acknowledgement from App.C that the event has been taken from the queue;
   e. The calling of the App B's synchronous Web Services from App.A with invalid data;
   f. The calling of the App B's asynchronous Web Service 2.
2. Using App.C or the capability of the SOA Backplane, demonstrate how App.C can unsubscribe the service (in other words, the App C's asynchronous Web Service will not be called).
3. Use App.A to call App B's Web Service again and show that the messages are no longer delivered to App.C.

3.2.5. *Output*

Test Report.

3.3. Logging and traceability of a Web Service

3.3.1. *Objective*

The objective of this scenario is to clearly show how all actions across the CCN2 Platform are logged to enable full traceability and auditing.

3.3.2. *Approach*

The approach is to run the Web Services and test what is logged. The approach must clearly show how non repudiation of Web Service calls (which are successfully completed) will be achieved.

3.3.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;

3.3.4. *Steps*

1. The test must demonstrate the following events are logged and are subsequently accessible for review:
   a. The Web Service calls;
   b. The response to the Web Service calls;
   c. The time and date the Web Service was called (i.e., entry time to the CCN2 Platform);
   d. The intermediate steps which were carried out (i.e. transformation);
   e. The authentication credentials which were provided;
   f. The logging of the authentication and authorisation requests to the IAM Module;
   g. The date and time the message was delivered to the adapter to invoke the business application;
   h. The date and time the response message was received from the business application;
   i. The date and time the response message was delivered to the requesting application (App.A);
   j. The date and time the message was delivered to the queue of the asynchronous Web Service;
   k. The data and time that the message was picked up from the queue by App.C.
2. Turn off certain aspects of logging (for example delivery to the queue and pick up by App.C), and re-run the Web Services to show that these events are no longer logged.

3.3.5. *Output*

Test Report.

3.4. Messages' Persistence and Recoverability

3.4.1. *Objective*

The objective of this scenario is to demonstrate the error handling, persistence of messages and recoverability when applications are off line and Web Services calls cannot be completed.

3.4.2. *Approach*

The approach adopted is to simulate the unavailability of specific applications and show how the CCN2 Platform Proof of Concept responds.

3.4.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;
3. Acceptance Test Plan.

3.4.4. *Steps*

1. Take App.B off line to simulate application or network failure;
2. Call the synchronous Web Service using App.A and show how the Web Service times out with a response to App.A that the service call could not be completed;
3. Restart App.B;
4. Take App.C off line to simulate application or network failure;
5. Show how the messages are persisted in a queue while App.C is off-line (clearly show the message being persisted);
6. Bring App.C back on line and demonstrate how recovery is carried out in terms of the messages being retrieved from the queue and the acknowledgement messages being generated once App.C has taken the messages;
7. Take App.B off line to simulate application or network failure;
8. Call App B's asynchronous Web Service 2 and show how the CCN2 Platform can queue the request;
9. Restore App.B and show how the asynchronous web call is completed;
10. Call App B's asynchronous Web Service 2 and take App.A off line before the response message can be delivered;
11. Demonstrate how the response message can be queued while App.A is off line;
12. Bring App.A back on-line and show how the persisted response message is delivered to App.A.

3.4.5. *Output*

Test Report.

3.5. Dynamic monitoring of a Web Service

3.5.1. *Objective*

The objective of this scenario is to show how the CCN2 Platform can be monitored in a dynamic manner.

3.5.2. *Approach*

The approach is to use the technical activity monitoring functionality of the CCN2 Platform to dynamically monitor Web Services in-flight.

3.5.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;

3.5.4. *Steps*

1. Set up continual calling of Web Service version 1.0 and version 1.1 from App.A;
2. Demonstrate via the activity monitoring console how the Web Service calls can be dynamically monitored so that there is a near real-time view in terms of what is happening on the SOA Backplane in terms of activity and resource usage.

3.5.5. *Output*

Test Report.

3.6. SOA Governance Policy Set up and Run Time Enforcement

3.6.1. *Objective*

The objectives are to:

1. Set up SOA run time governance policies and to test them in operation at run time;
2. Set up SOA build policies and to test them at deployment time.

3.6.2. *Approach*

The approach is:

1. To use the SOA Governance Module to set up specific run time policies which are testable at run time;
2. To use the SOA Governance Module to set up specific build policies which are testable at deployment;
3. Create policy violations to test policy enforcement;
4. To generate alerts on policy violations.

3.6.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;
3. Acceptance Test Plan.

3.6.4. *Steps*

1. Demonstrate how a policy can be set up so that when the synchronous Web Services are called more than a certain amount of times in a specified time period (i.e. more than 100 times per minute) an alert will be raised. Ensure that the thresholds for version 1.0 and version 1.1 of the Web Service differ;
2. Demonstrate how a policy can be set up limiting the number of times in a specified time period that the defined synchronous Web Service can be called before further calls are rejected and an alert is generated. Ensure that the thresholds for version 1.0 and version 1.1 of the Web Service differ;
3. Demonstrate how a policy can be set up where synchronous Web Service version 1.0 can only be called with the authentication details of specified users while Web Service version 1.1 can be called without any authentication;
4. Demonstrate how a policy can be set up where the synchronous Web Service version 1.0 will only be called if the service consumer has encrypted the message payload;

5. Demonstrate how a policy can be set up whereby an alert is generated if more than a certain number of Web Services are called with an invalid payload;
6. Demonstrate how a policy can be set up whereby the maximum message size is specified and any message above this size will be rejected;
7. Clearly show how the policies are not hardcoded and can be changed via parameters accessible via a GUI or other user interface;
8. The synchronous Web Services (version 1.0 and version 1.1) is called by App.A in order to exceed the defined thresholds (number of times in a specified time period plus maximum number of times). Generation of the alert must be demonstrated as well as the rejection of the call;
9. Use App.A to call synchronous Web Service version 1.0 without any authentication details and show the rejection of these requests;
10. Use App.A to generate Web Service calls where the message payload is not encrypted and the rejection of these requests must be generated;
11. Use App.A to generate Web Service calls with an invalid payload in order to exceed the defined thresholds and the generation of the alert must be demonstrated;
12. Use App.A to generate Web Service calls with a message payload which exceeds the defined size. The rejection of the request must be demonstrated;
13. Set up a SOA Governance build policy which clearly states certain naming conventions (for example a version number);
14. Modify one of the previous developed Web Services so that it no longer conforms to the build policy and attempt to deploy it on the SOA Backplane.

3.6.5. *Output*

Test Report.

3.7. Performance and Scalability

3.7.1. *Objective*

The objective of this scenario is to gain some understanding of the performance and scalability characteristics of the proposed CCN2 Platform configuration by injecting Web Services calls with varying payloads and characteristics.

The configuration used for the Proof of Concept is at the discretion of the Tenderer. The Tenderer may scale down (i.e. from the production configuration) to a smaller configuration. In this event, the volume of transactions should also be appropriately scaled down. However, the Tenderer must clearly indicate and justify how the Proof of Concept configuration can be scaled up to meet the required production volumes as stated in the non-functional requirements.

Please note that generic benchmark data is not acceptable as a substitute for this scenario although it may be provided as an appendix.

In all cases the following key metrics need to be recorded during the tests:

a. Number of Web Service calls' processes in the specified time intervals;
b. Average time to complete a Web Service call (note this must be within the boundaries of the CCN2 Platform) and it must be broken down by synchronous and asynchronous calls;
c. Average XML to XML transformation time per message size;
d. Average EDIFACT to XML transformation time per message size;
e. Encryption/decryption time per message size;
f. Authentication/authorisation time.

3.7.2. *Approach*

The approach is as follows:

1. Firstly, generate an average Web Services load with a mixture of message sizes as per the non-functional requirements;
2. Secondly, generate a peak Web Services load for thirty minutes as per the non-functional requirements;
3. Thirdly, generate a Web Services load which exceeds the peak load by a factor of two and for which the PoC is not designed to handle.

The following are some characteristics which must be demonstrated during the performance scenario:

1. A mixture of message sizes as follows:
   a. Small Messages – 5 KB
   b. Medium Messages – 100 KB
   c. Large Messages – 5 MB

   d. Extreme Messages – 1 GB

2. The percentages of each message size will be as per the volumes given in the non-functional requirements;
3. 5% of the messages will undergo EDIFACT to XML transformation;
4. 10% of the messages will be encrypted and will need to be decrypted before processing;
5. 5% of the messages will require XML to XML transformation;
6. Authorisation against the IAM Module for all Web Service calls using minimum 10 different credentials;
7. Synchronous Web Service calls 50% of the traffic;
8. Asynchronous Web Service calls 50% of the traffic.

3.7.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;

3.7.4. *Steps*

1. Generate the average load and record the performance;
2. Generate the peak load and record the performance;
3. Generate the stress load and record the performance.

3.7.5. *Output*

Test Report

3.8. IAM Module integration

3.8.1. *Objective*

The objective of this scenario is to show how the IAM Module is integrated with the CCN2 Platform to manage authentication of users and authorisation requests to access services.

3.8.2. *Approach*

The approach is to use the IAM Module to authenticate users and authorise access to specific services. One version of the synchronous Web Service specifically requires authentication against credentials held on the IAM Module before it can be accessed.

3.8.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;

3.8.4. *Steps*

1. Use App.A to call the synchronous Web Service version 1.0 and demonstrate the logging of the authentication request against the IAM Module and show how this user is authorised to use version 1.0 of the Web Service;
2. Change the authentication credentials within the IAM Module so that the provided authentication credentials no longer provide access to the defined Web Service;
3. Use App.A to call the Web Service again and demonstrate the rejection of the authentication request by the IAM Module and demonstrate that the Web Service has not been run but both the request and the rejection have been logged.

3.8.5. *Output*

Test Report.

3.9. Access Point Failover

3.9.1. *Objective*

A design principle for the CCN2 Platform requires a local access point. The functionality which is available on the Access Point is a Tenderer design decision. However one of the key purposes of the Access Point is to provide a level of resilience to ensure that the CCN2 Platform can cope with the unavailability of:

1. Network connectivity or;
2. The Main Hub or;
3. The Access Point.

The objective of this scenario is to demonstrate how the CCN2 Platform can cope with these events.

3.9.2. *Approach*

The approach is to simulate the following events to test how the CCN2 Platform will provide the required level of resiliency:

1. Network/Main Hub unavailability;
2. Access Point unavailability.

3.9.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;

3.9.4. *Steps*

1. Simulate the unavailability of the network links with App.B and App.C;
2. Use App.A to call the asynchronous Web Service 2 and demonstrate how the design copes with the unavailability of the network links by queuing the request;
3. Restore the network links and demonstrate how the queued requests are processed;
4. Simulate the unavailability of the Main Hubs;
5. Use App.A to call the Web Services and demonstrate how the design copes with the unavailability of the Main Hubs;
6. Restore the Main Hubs and demonstrate how the CCN2 Platform recovers, so that the local Access Point is now synchronised with the Main Hubs;
7. Simulate the unavailability of the Access Point and demonstrate how a different Access Point can be used to access Web Services.

3.9.5. *Output*

Test Report.

3.10. Platform Failover

3.10.1. *Objective*

Reliability is a key non functional requirement for the CCN2 Platform and any design must provide a high level of availability. It is envisaged that the Proof of Concept will have an active-active or active-passive configuration which provides quick recovery from any hardware or software failure. Therefore, the objective of this scenario is to demonstrate that the configurations can failover without loss of in-flight transactions.

3.10.2. *Approach*

The approach is to simulate the failure of an active hub and demonstrate the recovery capability.

3.10.3. *Inputs*

1. The Web Services and applications;
2. Test Design Specifications;

3.10.4. *Steps*

1. Set up a test system which continually calls the available Web Services from App.A (preferably in a peak load manner);
2. Simulate the immediate unavailability of an active node (for instance by cutting power) which is processing in-flight transactions;
3. Show how the overall PoC caters for the unavailability of an active node without loss of service – In other words show how a Web Service is called after the simulated failure is processed;
4. Show how the integrity of in-flight transactions are maintained when an active -node fails – In other words how is the integrity of in-flight transactions maintained.

3.10.5. *Output*

Test Report.

3.11. Master Data Management

3.11.1. *Objective*

The objective of this scenario is to demonstrate the master data management capabilities of the CCN2 Platform.

3.11.2. *Approach*

The approach is to create a federated master data set and show how it is synchronised and replicated to another slave data set.

3.11.3. *Inputs*

1. The master data sets as previously defined (see section 1.3);
2. MDM functionality of the CCN2 Platform;
3. Test Design Specifications;

3.11.4. *Steps*

1. Demonstrate that Organisation B cannot change master data which is controlled by Organisation A and vice versa;
2. Demonstrate how workflow can be used to approve changes to Organisation A's master data;
3. Demonstrate how publish/subscribe mechanisms can be used to notify organisations when master data has changed;
4. Demonstrate how a history of master data changes can be recorded with specific rules for the validity of different versions (i.e. time based);
5. Demonstrate how master data which can be changed by Organisation A is replicated (in an encrypted manner) to Organisation B and C;
6. Demonstrate how master data which can be changed by Organisation B is replicated (in an encrypted manner) to Organisation A and C;
7. Demonstrate how a full update of the master data set can be replicated to Organisation C (in an encrypted manner) on a request from Organisation C;
8. Demonstrate how the federated master data set can be altered so that a master data element previously controlled by Organisation A is not controlled by Organisation B;
9. Demonstrate how authentication and non repudiation can be supported when master data is being replicated.

3.11.5. *Output*

Test Report.

End of ANNEX II.D