



Networking for Communications Challenged Communities:
Architecture, Test Beds and Innovative Alliances
Contract no: 223994

D7.2 N4C Laboratory Testing on Integrated Subsystems



Pedro Nunes Institute

António Cunha, André Pardal, Francisco Barbosa
Postal address: Rua Pedro Nunes, 3030-199, Coimbra, Portugal
Tel: +351 239700934
Fax: +351 239700912
www.ipn.pt
cunha@ipn.pt

ABSTRACT (Max 400 word)

Starting in May 2008, N4C is a 36 month research project in the Seventh Framework Programme (www.cordis.lu/fp7). In cooperation between users in northern Sweden and Kočevje region in Slovenian mountain and partners, the project will design and experiment with an architecture, infrastructure and applications in field trials and build two test beds.

This document makes reference to some of the actual integration methodologies, explaining the advantages and disadvantages of each one, along with the first drafts for the tests that will be performed in the summer of 2009.

Due date of deliverable: 30/04/2009 Actual submission date: 30/04/2009

Document history		
Status	Date	Author
Initial Draft based on DOW	15-04-2009	António Cunha, André Pardal, Francisco Barbosa
First draft circulated to consortium	16/04/2009	
Feedback	23/04/2009 18/06/2009	Karl Grøttum Maria Udén
Submission to EC		

Dissemination level	
	Level
PU = Public	PU
PP = Restricted to other programme participants (including the Commission Services).	
RE = Restricted to a group specified by the consortium (including the Commission Services).	
CO = Confidential, only for members of the consortium (including the Commission Services).	

CONTENT

1. INTRODUCTION	4
1.1 OVERVIEW	4
1.2 ABBREVIATIONS	4
2. SYSTEM'S INTEGRATION	4
2.1 SYSTEM'S GENERAL DIAGRAM	5
2.2 SYSTEM'S FUNCTIONALITIES	6
2.3 MODULES' DESCRIPTION	7
2.3.1 APPLICATION'S SWIM LANE	7
2.3.2 TRANSPORT SWIM LANE	10
2.3.3 NETWORK SWIM LANE	11
2.3.4 LINK AND PHYSICAL SWIM LANES	12
2.3.5 PHYSICAL DEVICES	12
3. TEST METHODOLOGY (THEORETICAL FRAMEWORK).....	12
3.1 UNIT TESTS	13
3.2 INTEGRATION TEST	13
3.2.1 GLOBAL INTEGRATION TESTING	13
3.2.2 INCREMENTAL INTEGRATION TESTING	14
3.2.3 INTER-PROCESS COMMUNICATION (IPC)	16
3.3 FUNCTIONAL TESTS	17
3.4 NON-FUNCTIONAL TESTS	17
4. REQUIREMENTS-BASED TESTING.....	18
5. SUBSYSTEMS' INTEGRATION FOR SUMMER TESTS	19
5.1 SYSTEM INTEGRATION	20
5.2 SUBSYSTEM 1 – TRANSMISSION	20
5.3 SUBSYSTEM 2 – DISCOVERY & LINK	21
5.4 SUBSYSTEM 3 – BUNDLE & STORAGE	22
5.5 SUBSYSTEM 4 – DATA MANAGEMENT	23
5.6 SUBSYSTEM 5 – SYSTEM MANAGEMENT CONFIGURATION	24
5.7 SUBSYSTEM 6 – ROUTING	25
6. SUBSYSTEM TEST PLANNING	26
6.1 LINK AND PHYSICAL SWIM LANES	26
6.1.1 WI-FI	26
6.1.2 WIMAX	27
6.1.3 BLUETOOTH	27
6.2 TRANSPORT SWIM LANE	27
6.2.1 PROPHET	27
6.2.2 DTN BUNDLE PROTOCOL	28
6.3 APPLICATION SWIM LANE	28
6.3.1 EMAIL	28
6.3.2 NOT SO INSTANT MESSENGER	28
6.3.3 WEB CACHING	29
6.3.4 HIKER'S APPLICATIONS	34
6.3.5 HERDER APPLICATION AND ANIMAL MIGRATION MONITORING	36
6.3.6 MANAGEMENT AND CONFIGURATION CONTROLLER	37
7. CONCLUSIONS.....	37
8. REFERENCES	38

1. INTRODUCTION

1.1 OVERVIEW

This document was created under the scope of the preparation of the system's integration (Work package 7).

For this document, it was planned the development and supervision of incremental lab testing of the integrated hardware and software and also the development of a regression harness for testing updated infrastructure subcomponents. However, due to the project's complexity, it is not yet possible to design a complete set of integration tests in order to ensure the total correctness of the system's integration. To achieve this task with the maximum efficiency, we have decided to make a theoretic approach first, which is a fundamental requirement for the next step in the development process, the integration tests design.

So, in the following sections, it is analyzed the division of the system into modules, grouped according to each one's function, which are then described. Later, different integration test methodologies are analyzed against the objectives and constraints of this project, explaining the reasons why some of them fit into this project and others do not.

Finally, it is proposed a division into subsystems as part of the methodology used for the integration tests that will occur next summer, along with conditions that should be established before initiating tests.

The document aims to not only expose the theory for integration tests, selecting the solutions that seem to be more appropriate, but also to help N4C's partners to develop their white-box tests, giving them, at the same time, a general view of the other modules and the entire system itself.

1.2 ABBREVIATIONS

CCR	Communications Challenged Region
DTN	Delay Tolerant Networking
IPC	Inter-Process Communication
LI	Legacy Internet

2. SYSTEM'S INTEGRATION

System's integration is the result of joining all the modules of the system, ensuring that they properly work together as a system. Thinking of a system as a group of, many times disparate, subsystems, each of which with its own interface, the integration of the subsystems has to do with how the interfaces will connect to each other, trying to achieve a way of all interfaces can communicate between them, so all subsystems can act as one.

Integrated systems can, in its turn, be part of a bigger system if another integration process is made, leading to every time more complex systems, but at the same time, to a value-adding capability.

2.1 SYSTEM'S GENERAL DIAGRAM

Using gathered information from N4C's partners, we have defined the modules of the system. From the interaction of the modules it was build the system's general diagram presented in Figure 1[4]. This diagram is divided in six swim lanes, five of which correspond to the communication's stack and the sixth to other hardware with different functions, not restricted to network capabilities.

In this diagram there are two kinds of paths between the modules: the control message path and the network message path. Each type of message can travel by only one of the paths:

1. Control message: This type of message is related to, among others, system configurations messages, notification messages and link state messages. Only control messages can travel in the red paths, as depicted below;
2. Network message: Data messages from/for user's applications, which may also include control data, travels in the blue paths.

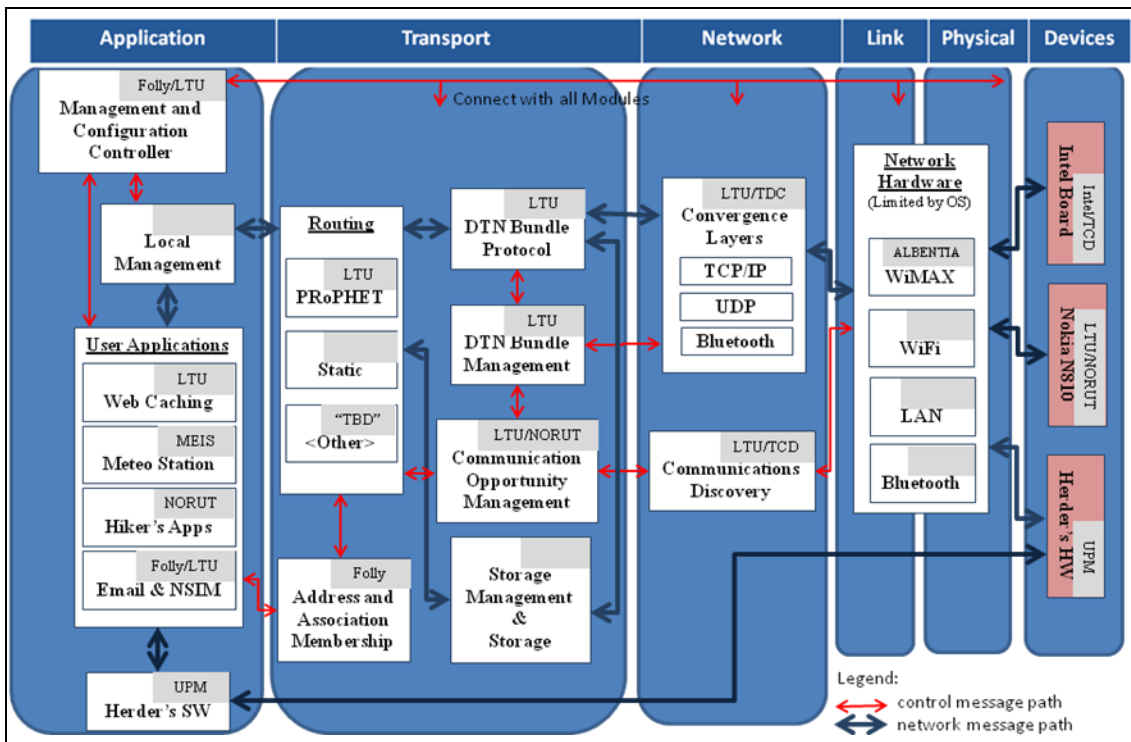


Figure 1: System's general diagram

2.2 SYSTEM'S FUNCTIONALITIES

The following UML use case diagram presents the functionalities that N4C intends to offer its users, divided by the subsystems discussed next in section 2.3.1.

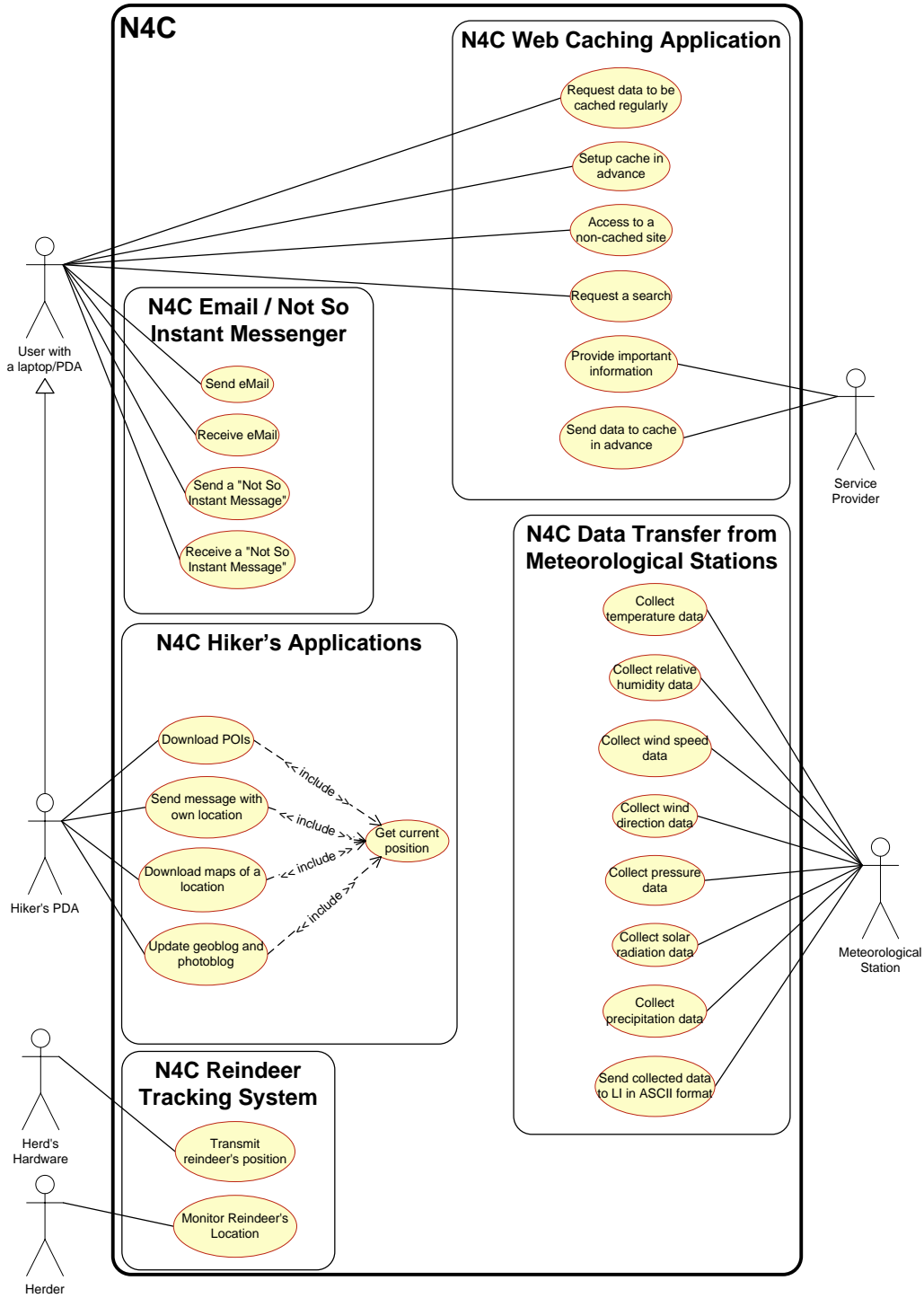


Figure 2: Functionalities use case diagram.

2.3 MODULES' DESCRIPTION

2.3.1 APPLICATION'S SWIM LANE

2.3.1.1 USER APPLICATIONS

The user applications will be the interface of the system with the user. This module is the source/destiny of the data that will be generated. There are four user applications, namely web caching, data transfer from meteorological station, hiker's applications and email/not so instant messenger.

2.3.1.1.1 WEB CACHING

This application has the following functionalities:

- User pulled regular: a user can request some data to be delivered regularly;
- User pulled event driven: a user can set up a cache in advance;
- Provider pushed regular: Provide important information for educational purposes and tourists, etc.;
- Provider pushed event driven: Sends data to cache in advance;
- Ad hoc site request: a user tries to access a site that is not normally cached;
- Ad hoc search request: a user makes a search request through a search box.

This application should have the following input: site/search requests from clients using ad hoc TCP/IP mode in the CCRs and data from providers using TCP/IP in the LI. The answer to the input should be the delivery of the requested data to the client.

In Figure 3, the CCR area should be viewed as a black box, being the DTN users and service providers the ones who provide the input, and the DTN users are the output's destination.

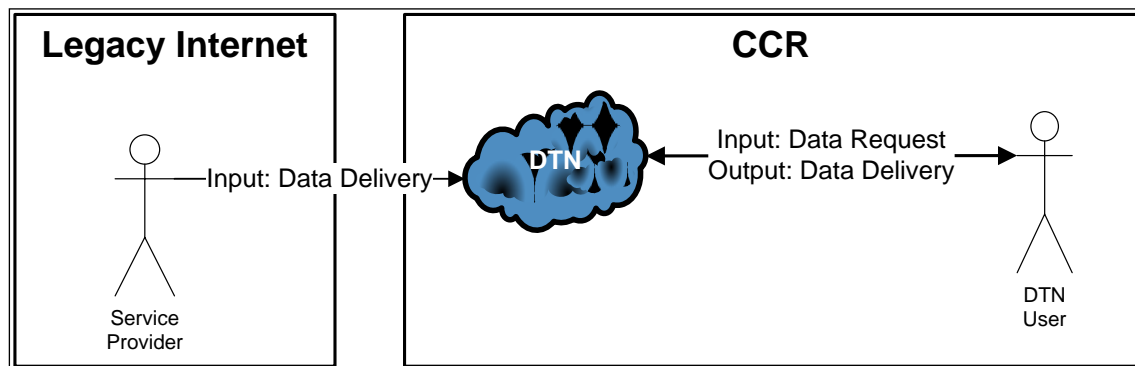


Figure 3: Input and output of Web Caching application

2.3.1.1.2 METEOROLOGICAL STATION

This application transmits meteorological data from the DTN to LI. The meteorological sensors measure:

- a) the temperature;
- b) the relative humidity;
- c) wind's speed and direction;
- d) the pressure;
- e) the solar radiation; and
- f) the precipitation.

Before of data's transmission, the data file should be prepared so the following initializations can be done:

- init file for meteorological station, defining the constants for the sensors;
- init file for meteorological station, defining the processing of the measured data;
- init file for meteorological station, defining the data to be transmitted;
- init file for the DTN protocol defining the destination of data.

The data files should be delivered in ASCII format (see D3.1 Functional Specification (Initial), document: N4C-WP3-2-fs-initial-04.doc).

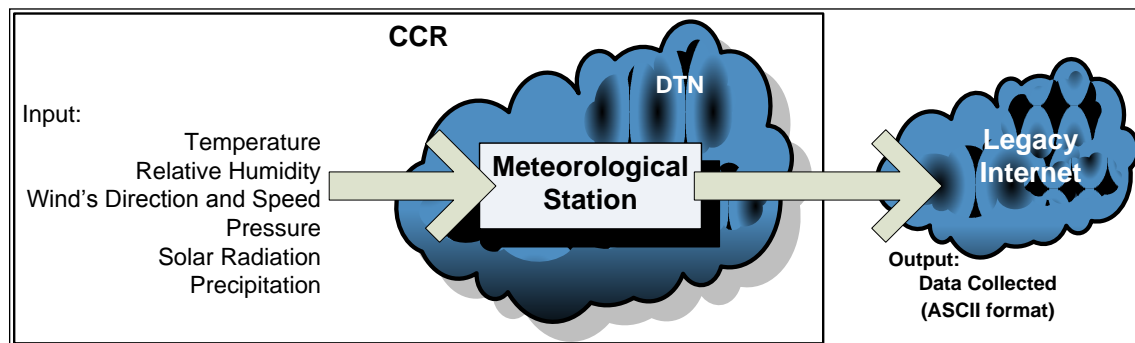


Figure 4: Input and output of the meteorological station

2.3.1.1.3 EMAIL / NOT SO INSTANT MESSENGER

This application has two distinct functionalities. The first one is to send and receive emails, and the latter is “*not so instant messages*” (similar to Microsoft Windows messenger). Both of these functionalities can use text or attachments. The “*not so instant messenger*” will be implemented within the DTN network only to guarantee the correct work of this functionality.

The inputs of these applications are the emails/messages sent out by the “DTN users” and the output is the delivery of emails/messages to the destination.

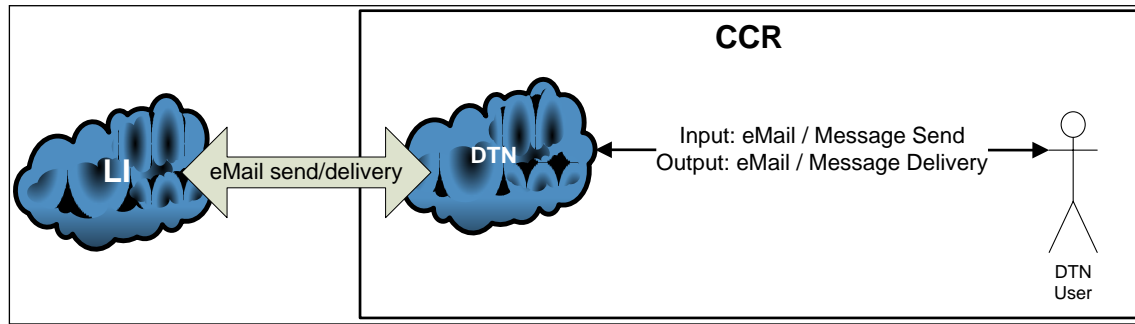


Figure 5: Input and output of the email/Not So Instant Messenger Application

2.3.1.1.4 HIKER'S APPLICATIONS

This module consists in several features, as described below:

- **Points of Interest (POI)** – The functionalities of the application are to download potential interest points like heart starter, medical or physical services. The module should be able to extract GPS maps with the location of interest points, from the web cache/internet. For example, the location of nearest medical services (medical, physical or heart starter) can be one type of Point of Interest (POI). The input of the application is the current location, given by the GPS receiver's coordinates and the output is the POI database updated.
- **Send a Message with own location** – This application should be able to send a message with GPS receiver's own location in a DTN region and forward it to the LI. The input of this application is the GPS receiver's coordinates and the output is the email containing those coordinates.
- **Maps for Own Location** – Download maps related to one's location. The application should be able to pull up a map of the area around the current GPS receiver's coordinates. The input of this application is the GPS receiver's coordinates and the output is the updated cached maps.
- **Geoblog and Photo Blog** – Must be able of automatic updates of geoblog and photo blog when there is a connection opportunity. The input of this functionality is the GPS receiver's coordinates and the output is the email with the GPS receiver's coordinates.

These features are schematized according to its input and output in the diagram depicted in Figure 6.

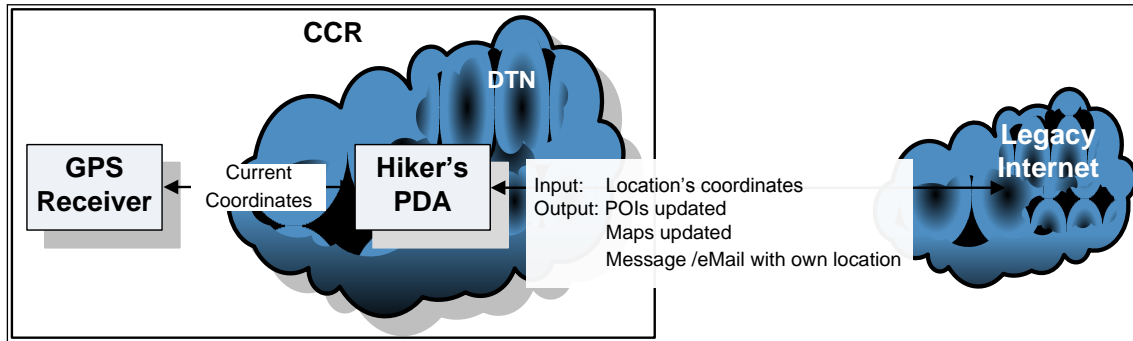


Figure 6: Input and output of Hiker's Application

2.3.1.2 HERDER'S SOFTWARE

This module represents a specific subsystem focused in the animals monitoring, connected to a specialized wireless hardware.

2.3.1.3 MANAGEMENT AND CONFIGURATION CONTROLLER

The management and configuration controller module is connected with all other modules and will ensure the configuration, monitoring and analysis of all the system's modules.

2.3.1.4 LOCAL MANAGEMENT

The local management module implements the interface between user applications and the remainder system and manages data from/to user applications.

2.3.2 TRANSPORT SWIM LANE

2.3.2.1 ROUTING

This module manages and calculates the transition's probability of the bundles between two nodes. For now, there are two kinds of routing protocols that can be implemented: the PROPHET and the static protocol.

2.3.2.1.1 PROPHET

The PROPHET uses three main parameters: alpha, beta and gamma. The alpha parameter is used for updating the deliverable probability of the encountered node, beta is used for updating the transitive deliverable probabilities and gamma is used for aging the deliverable probabilities.

2.3.2.2 ADDRESS AND ASSOCIATION MEMBERSHIP

Allows the association of a new member in the CCRs and manage nodes' addresses.

2.3.2.3 DTN BUNDLE PROTOCOL

This module will bundle/debundle data.

2.3.2.4 DTN BUNDLE MANAGEMENT

This module is responsible for creating, managing and breaking links between two neighbor nodes.

2.3.2.5 COMMUNICATION OPPORTUNITY MANAGEMENT

This module manages the connection between nodes. When a new neighbor becomes accessible, this module performs the following steps:

- a) Decides what routing protocol should be used;
- b) Wakes up the routing protocol;
- c) Exchanges the “meta information” (link characteristics: how long it expects the link to be up and how fast data can be exchanged) with the new neighbor;
- d) Decides what bundles to exchange;
- e) Order the bundle protocol to start exchanging bundles.

2.3.2.6 STORAGE MANAGEMENT & STORAGE

This module stores and manages the bundles. It evaluates what bundles need to be send first and how much storage capability is available.

2.3.3 NETWORK SWIM LANE

2.3.3.1 CONVERGENCE LAYERS

This module converts the bundles in conventional legacy internet packets and converts the packets from conventional legacy internet into bundles.

2.3.3.2 COMMUNICATIONS DISCOVERY

This module searches for other nodes using UDP packets sent by wireless hardware in broadcast mode and waits for an answer.

2.3.4 LINK AND PHYSICAL SWIM LANES

2.3.4.1 NETWORK HARDWARE

This module represents the physical devices which will send and receive data.

2.3.5 PHYSICAL DEVICES

2.3.5.1 INTEL BOARD

It is a specialized hardware set (batteries, data store and so on) developed to work within DTNs and with arctic temperature's conditions.

2.3.5.2 NOKIA N810

It is an internet tablet. It features the Maemo Linux distribution, based on Maemo 4.0, including MicroB, a Mozilla-based mobile browser, and a GPS navigation application.

2.3.5.3 HERDER'S HARDWARE

This hardware is a wireless animal tracking system. Animal Tracking intends to develop novel sensors and power sources to allow reindeer and other herds/cattle to be located within the DTN region. The air-interface technologies will accordingly proceed to find optimal physical layer strategies.

3. TEST METHODOLOGY (THEORETICAL FRAMEWORK)

It is possible to define two basic opacity classes of tests: black box testing and white box testing. The black box testing consists in ignoring the internal mechanism of a system and focuses solely on the outputs generated in response to selected inputs. It is often used in validation tests. The white box tests are focuses on the internal structure of the module and is often used for verification.

The classes of testing are denoted by colors to depict the opacity of the testers of the modules. In the black box tests, the internal structure of the module isn't relevant for the test. The module is considered to be a black box to the tester who can't see inside the box. The tester knows only that information can be input into to the black box, and the black box will send something back out. Based on the requirements knowledge, the tester knows what to expect the black box to send out and tests to make sure the black box sends out what it is supposed to send out. In the white box tests, tests focus on the internal structure of the code

and are most of the time written by the code's developers, who perform white box tests by executing methods with certain parameters.

Our objective is to understand all the modules from a high level view only, due of the big complexity of the entire system. For that purpose, we will consider the modules like black-boxes. Afterwards, we expect that white-box tests (unit tests) are performed by the developers (each module owner testing its modules).

3.1 UNIT TESTS

Unit test is a method of software testing that verifies that the individual modules are working properly. Using white box testing techniques, testers can realize that a module does what it is intended to do at a very low structural level. Ideally, each test case is independent from the others. Additional components (like stubs) can be used to assist testing a module in isolation. For example, the tester will write some test code that will call a method with certain parameters and will ensure that the return value of this method is as expected. The goal of unit testing is to isolate each part of the system and show that the individual parts are correct. The unit test provides the guarantee that the system's module works correctly, allowing also to find problems in an early stage of the development cycle.

3.2 INTEGRATION TEST

Even though modules can be properly working individually, that does not mean that they can work integrated in a larger system or subsystem. For example, in a subsystem composed by multiple modules, data might get lost across an interface, messages might not get passed properly or interfaces might not be implemented as specified. In order to solve issues that might be encountered when integrating modules, an integration test must be performed.

The integration test is the phase of tests in which individual modules are combined and tested as subsystems. Then, testers apply the tests defined in an integration test plan to those subsystems.

The integration test can be divided into four methods: the global integration test, the incremental integration test and the Inter-Process Communication (IPC).

3.2.1 GLOBAL INTEGRATION TESTING

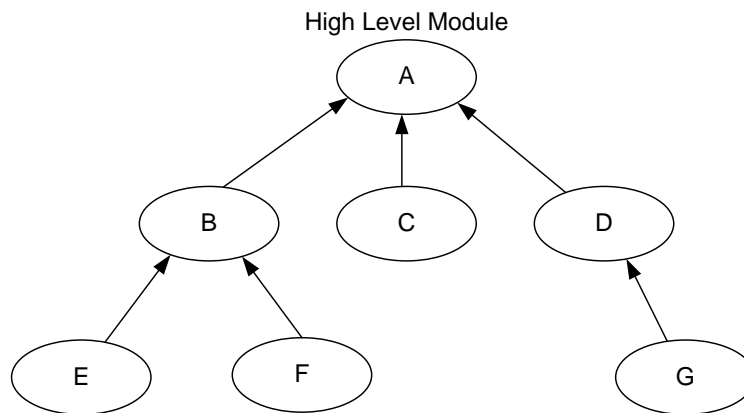
The global integration testing, or the big bang approach, consists in the integration of all modules at the same time. In this method, all modules are coupled together to form a complete system and then used for integration testing. The big bang method is very effective for saving time in the integration testing process, but the location of bugs can be very difficult after the bang, principally in complex systems.

3.2.2 INCREMENTAL INTEGRATION TESTING

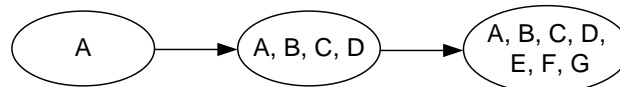
The incremental integration testing consists in incrementing the set of integrated modules progressively. This method results in some additional overhead relatively to the big bang method, but significantly reduces bugs' localization and time spent looking for bugs, particularly in complex projects. The two main techniques to perform incremental integration testing are the top-down and the bottom-up.

3.2.2.1 TOP-DOWN

The top down method is the procedure where the top integrated modules are tested and the branches of the modules are testing step by step till all modules are integrated. In this method the high level modules are tested first, possibly with the middle level control structures present only as stub (stubs are units which are only present to allow the higher level control routines to be tested). For example, a system with the following general modules diagrams:



The respective scheme of integration test using top-down technique is the following:



The most important advantages of this technique are:

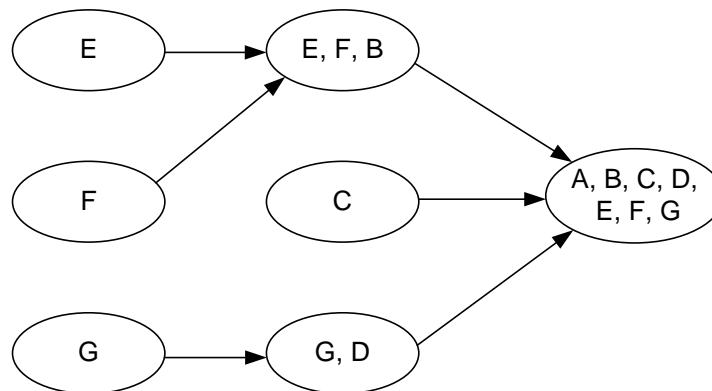
- allows early verification of high level behavior;
- modules can be added one at a time with each step if desired;
- it is easy to find the missing branch links;
- few or no drivers needed (unit to simulate the highest level modules).

The most important disadvantages are:

- delays verification of low-level behavior;
- stubs are required for missing elements;
- test cases are difficult to formulate and interpret.

3.2.2.2 BOTTOM-UP

In the bottom-up method the low level modules are integrated and tested first. After the integration testing of lower level integrated modules, they are combined in subsystems and these are tested. This process continues until the high level modules are tested. For the previous example, the scheme of integration test using bottom up technique is the following:



The main advantages of this method are:

- allows early verification of the low level behavior;
- no stubs are required;
- test cases more easily to formulate and interpret;
- it is easy to find bugs;
- logic modules tested thoroughly;
- testing can be done in parallel with implementation.

The principal disadvantages are:

- delays verification of the high level behavior;
- drivers are required for missing elements;
- possibility of a large number of modules integrated in the case that a module may have several modules' dependencies.

3.2.3 INTER-PROCESS COMMUNICATION (IPC)

IPC consists in a set of techniques for the exchange of data among multiple threads in one or more processes which may be running on one or more computers connected by a network, depending of the system. IPC techniques are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC). The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being exchanged.

An interesting IPC which we can use to integrate the system is D-Bus, a medium for local communication between processes running on the same host. D-Bus is meant to be fast and lightweight and is designed for use as a unified middleware layer underneath the main free desktop environments. Unlike more heavyweight conventional messaging middleware, D-Bus is non-transactional. It is stateful and connection-based, however, making it "smarter" than low-level message-passing protocols such as UDP. On the other hand, it does carry messages as discrete items – not continuous streams of data as in the case with TCP. Both one-to-one messaging and publish/subscribe communication are supported. D-Bus has a structured view of the data it carries, and deals with data in binary form: integral numbers of various widths, floating-point numbers, strings, compound types, and so on. Because data is not just "raw bytes" to D-Bus, messages can be validated and ill-formed messages rejected. In technical terms, D-Bus behaves as an RPC mechanism and provides its own marshaling. In Figure 7 [1] it is presented the D-Bus structure.

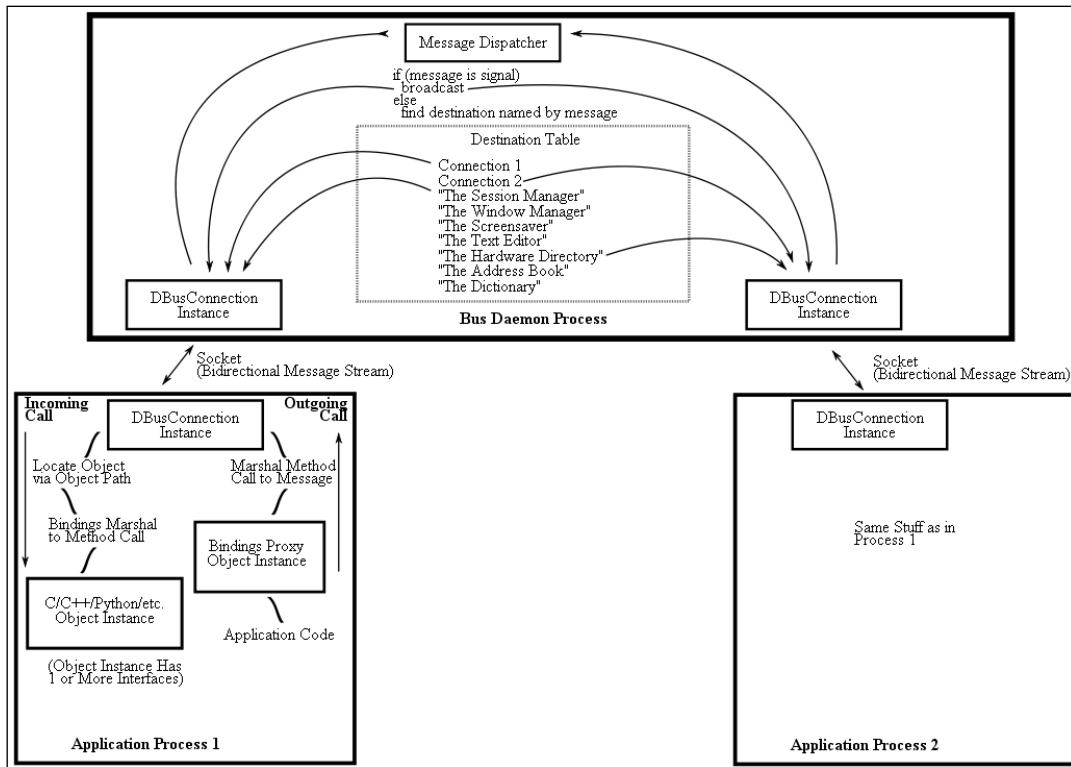


Figure 7: D-Bus structure.

3.3 FUNCTIONAL TESTS

The functional test has the aim of testing the complete integrated system to evaluate the system's compliance with its functional requirements. This kind of test is implemented within the scope of black-box testing, and as such, should require no knowledge of the internal structure of the modules.

The black-box approach is a testing method in which test data is derived from the specified functional requirements without regard to the final system's structure. It is also termed data-driven, input/output driven or requirements-based testing. Black-box testing also mainly refers to functional testing. It is a testing method emphasized on executing the functions and examination of their input and output data. The tester treats the module under test as a black-box. Only the inputs, outputs and functional requirements are visible, and the functionality is determined by observing the outputs to corresponding inputs. In testing, various inputs are exercised and the outputs are compared against functional requirements to validate its correctness. All test cases are derived from the requirement list.

3.4 NON-FUNCTIONAL TESTS

Non-functional testing verifies that the system functions properly, even when it receives invalid or unexpected inputs. Non-functional tests try to establish whether the device under

test can tolerate invalid or unexpected inputs, thereby establishing the robustness of input validation routines as well as error-handling routines. The most important non-functional tests are:

- Stress testing: conducted to evaluate the system at or beyond the limits of its requirements;
- Performance testing: conducted to evaluate the compliance of the system with specified performance requirements (capability, speed, precision, ...);
- Usability testing: conducted to evaluate the extent to which a user can learn to operate, prepare inputs for, and interpret outputs of the system;
- Load tests: maximize the load imposed on the system (volume of data, number of users, ...);
- Security tests: evaluates system's characteristics that relate to the availability, integrity and confidentiality of system's data and services;
- Recovery tests: subject a system to losses of resources in order to determine if it can recover properly from those losses;
- Reliability tests: is the probability that a system will operate without failure under given conditions for a given interval (mean time between failures = mean time to failure + mean time to repair).

4. REQUIREMENTS-BASED TESTING

We have decided to follow the requirements-based testing methodology that is addressed to validate that the requirements are correct, complete, unambiguous, and logically consistent and designing a necessary and sufficient (from a black box perspective) set of test cases from those requirements.

As this is an investigation-based project, aiming at the deployment of applications to final users, it is necessary to have test and integration methods, so that, in the end of the project, the system can be composed of tangible assets.

The requirements-based testing process can to be divided into the following steps:

- **Pass/Fail Criteria Definition** – The test effort has specific, quantitative and qualitative goals. Testing is successfully completed only when the goals have been reached. (e.g., testing is considered to be successfully completed if 100% of the messages are delivered correctly to its destination).
- **Design Test Cases** – Logical test cases are defined by five characteristics: the initial state of the system prior to executing the test, the data in the data base, the inputs, the expected outputs, and the final system state.
- **Build Test Cases** – To build the test cases we need: initialization, assumptions and restrictions identification. Creating the necessary data and building the components to support testing (e.g., build the necessities stubs and drivers).

- **Execute Tests** – Execute the test-case steps against the system being tested and document the results.
- **Verify Test Results** – Verify that the test results are similar to the expected ones.
- **Verify Test Coverage** – Track the amount of functional coverage and system coverage achieved by the successful execution of the set of tests.
- **Manage and Track Defects** – Any defects detected during the testing process will be tracked to resolution. Statistics are maintained concerning the overall defect trends and status.
- **Manage the Test Library** – The test manager maintains the relationships between the test cases and the programs being tested. The test manager keeps track of which tests have or have not been executed, and whether the executed tests have been passed or failed.

5. SUBSYSTEMS' INTEGRATION FOR SUMMER TESTS

After performing an evaluation of the four integration test possibilities described under section 3.2, we excluded the big bang approach and the top-down. The big bang approach was rejected because the modules are being developed progressively and the big bang approach requires that all modules are integrated at the same time, which would imply that the system's integration would have to wait until all the modules were fully developed. The top-down technique was rejected because it requires firstly the integration of the high level modules and, in this project, these modules will probably be the last to be terminated, because they need the cooperation of the end users in some specifications, like users' requirements. Consequentially, it is not logical to use this method.

The bottom-up method is a hypothesis, due to the fact that the low level modules will be the terminated first, which is consistent to the fact that the bottom-up method integrates first the low level protocol. Another advantage of this method is the great capability of localizing bugs in the integrated system.

Finally, IPC, namely the D-BUS integration platform, is also a hypothesis, as it adds flexibility and independency to the integration process: flexibility because it allows a faster accommodation of existing and new closed modules and independency of the integrated modules relatively of the other modules of the system. In this project those two features will be very important because the modules will be closed-in progressively.

After careful consideration, we believe that the D-Bus integration platform is the most interesting option, because D-Bus is fast, lightweight and is designed for use as a unified middleware layer underneath the main free desktop environments.

The bottom-up method was also excluded because D-Bus gives more flexibility in the integration process and increases the expansion capability, by adding new modules to the system.

5.1 SYSTEM INTEGRATION

The D-bus integration platform is an inter-process communication system which it can be used to integrate the modules of the system. Adding the D-bus system bus to the general diagram presented in Figure 1 resulted in the diagram depicted in Figure 8 [4].

Extrapolating from the diagram below, we've manage to define six subsystems as guidelines for the next summer tests, presented in the next sections.

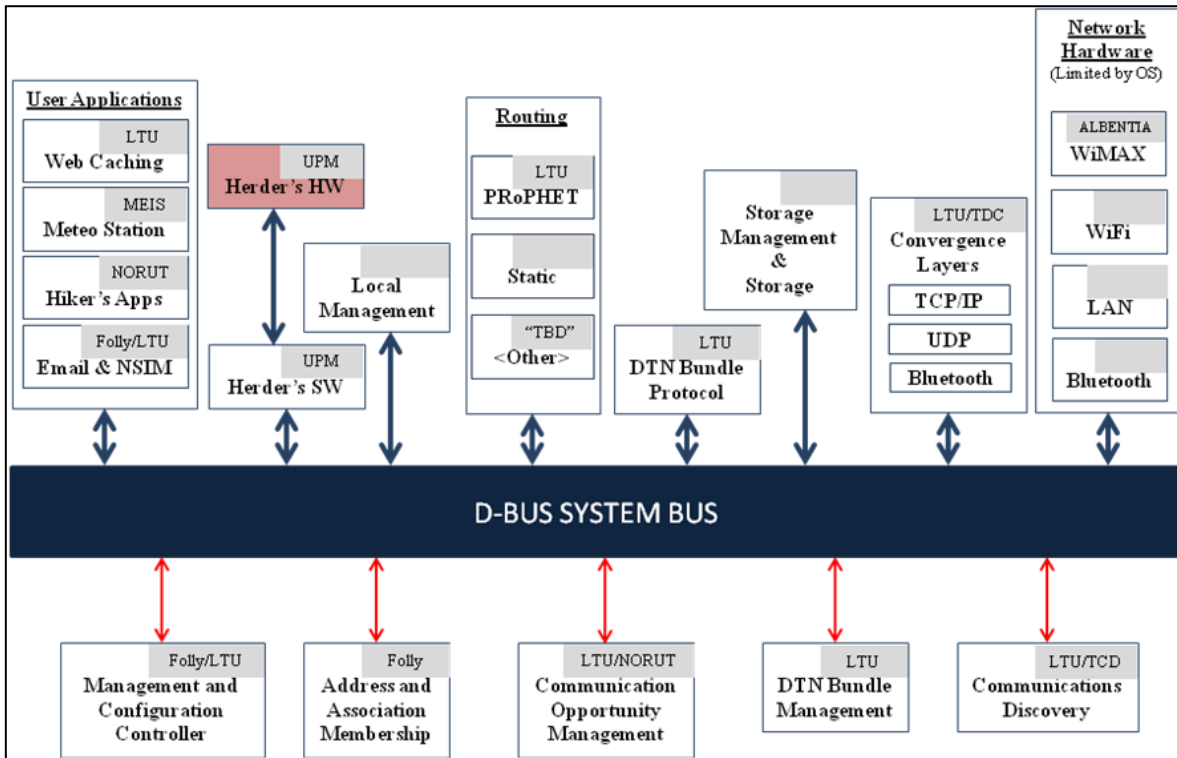


Figure 8: System general diagram using D-bus system bus.

5.2 SUBSYSTEM 1 – TRANSMISSION

The first subsystem, named Transmission, is emphasized in Figure 9 [4] and aggregates three modules: DTN Bundle Protocol, Convergence Layers and Network Hardware. The aim of this subsystem is to test the capability of transmitting data bundles using the available technologies.

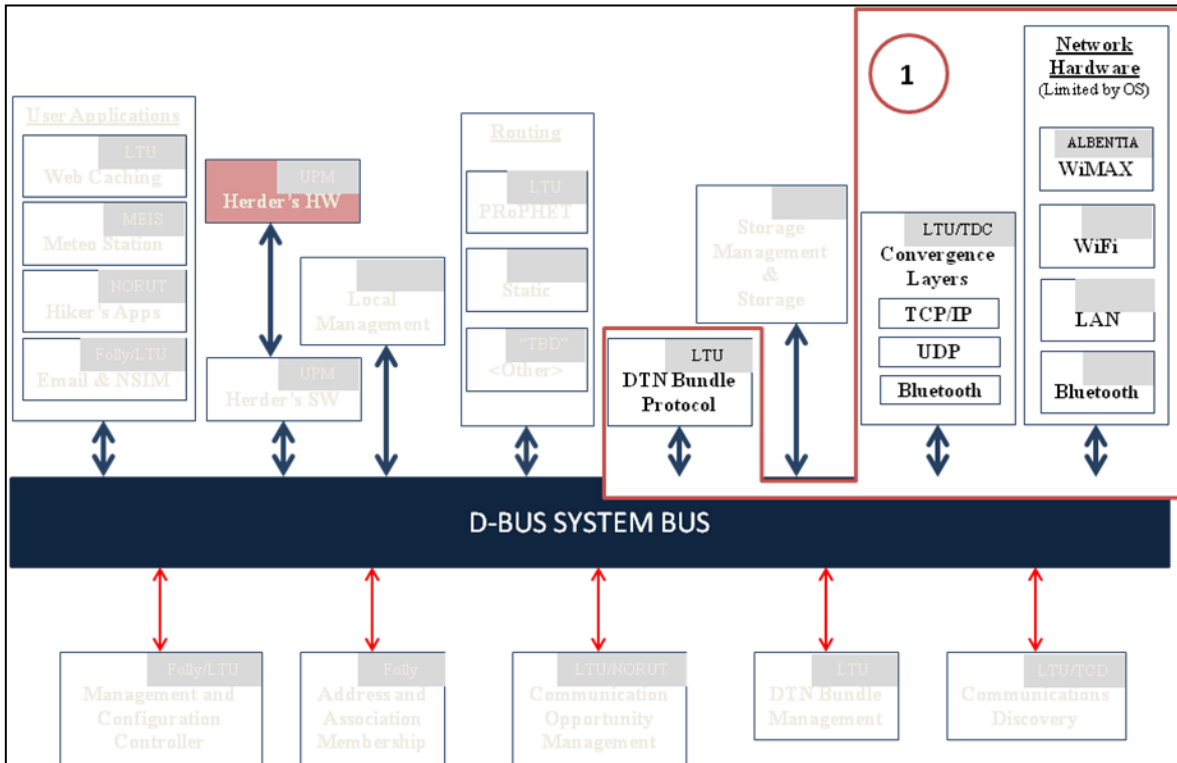


Figure 9: Definition of the subsystem 1 – Transmission – in the general diagram.

5.3 SUBSYSTEM 2 – DISCOVERY & LINK

Subsystem 2, named Discovery & Link, is comprised of five distinct modules: Communication Opportunity Management, DTN Bundle Management, Communication Discovery, Convergence Layers and Network Hardware. The aim of this subsystem is to test the capability of discovering new nodes, evaluate the conditions to establish a connection between two nodes and, if there are acceptable conditions, establish the link. This subsystem's constituent modules are emphasized in Figure 10 [4].

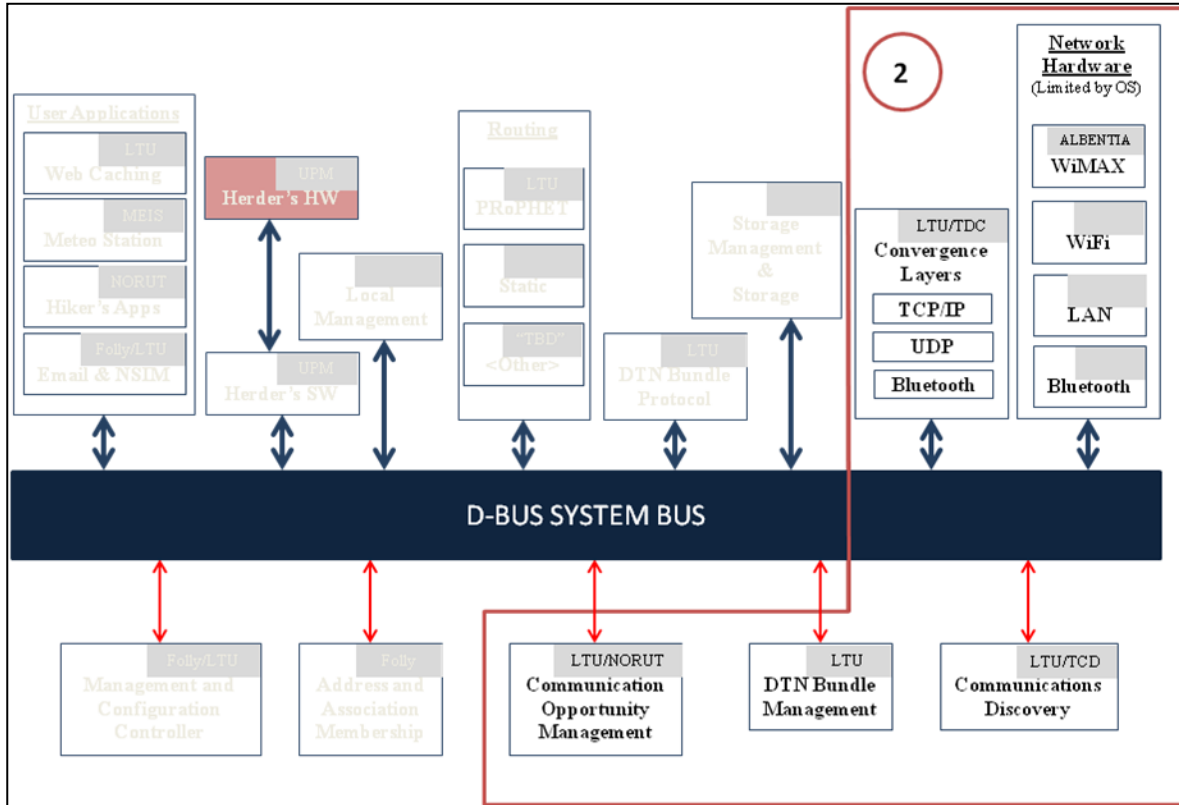


Figure 10: Definition of the subsystem 2 – Discovery & Link – in the general diagram.

5.4 SUBSYSTEM 3 – BUNDLE & STORAGE

Subsystem 3, named Bundle & Storage is composed by three modules: Routing, DTN Bundle Protocol and Storage Management & Storage. The aim of this subsystem is to test the functionalities of data bundle/debundle, data storage and storage management. In Figure 11 [4] it is outlined this subsystem’s position in the general diagram.

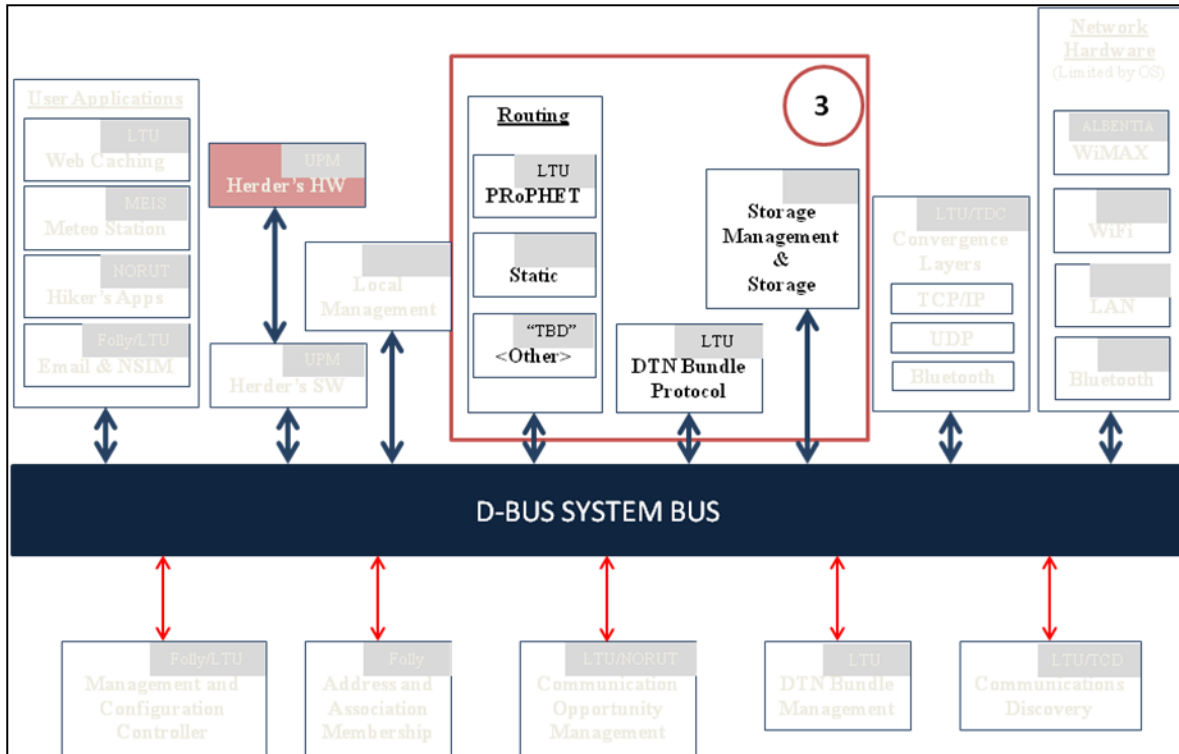


Figure 11: Definition of the subsystem 3 – Bundle & Storage – in the general diagram.

5.5 SUBSYSTEM 4 – DATA MANAGEMENT

Subsystem 4, named Data Management is composed by four modules: User Applications, Local Management, Herder’s Software and Herder’s Hardware. This subsystem is focused in the data generation and management of that data. Figure 12 [4] depicts this subsystem’s place in the general diagram.

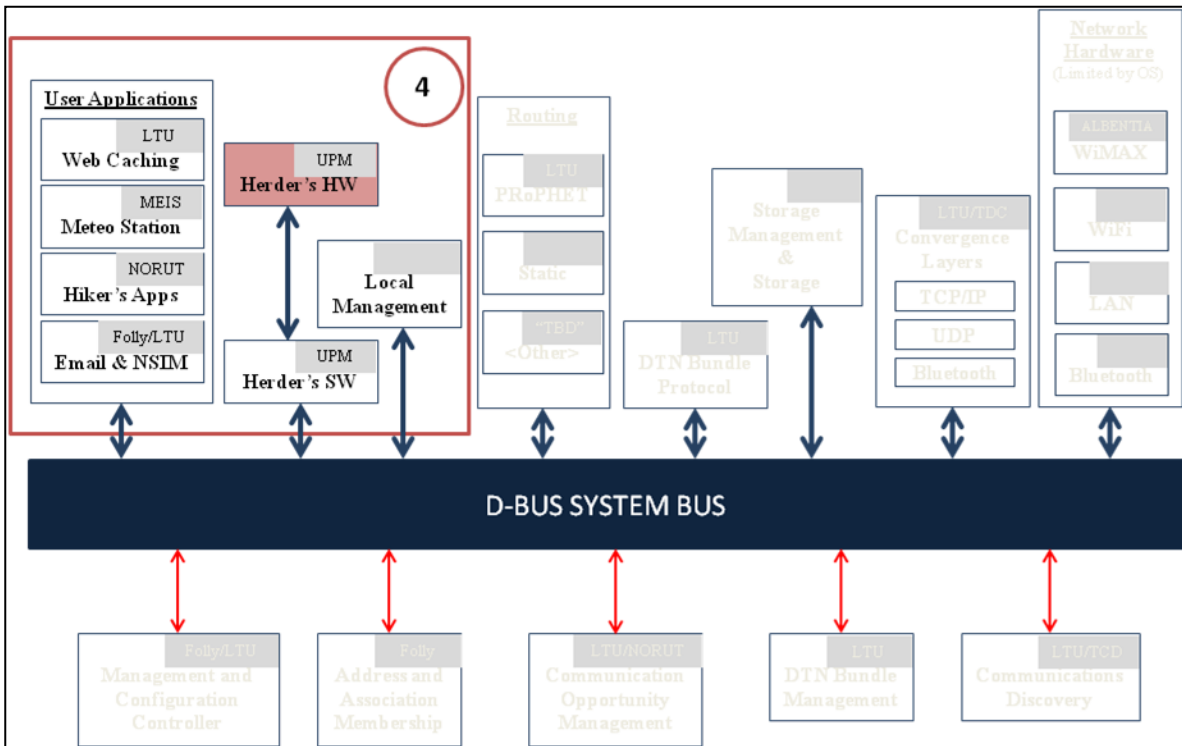


Figure 12: Definition of the subsystem 4 – Data Management – in the general diagram.

5.6 SUBSYSTEM 5 – SYSTEM MANAGEMENT CONFIGURATION

Subsystem 5, named System Configuration, as can be seen in Figure 13 [4], joins four modules: User Applications, Management and Configuration Controller, Herder’s Software and Herder’s Hardware. This subsystem is focused in the management and configuration’s test of the modules (e.g. User Applications).

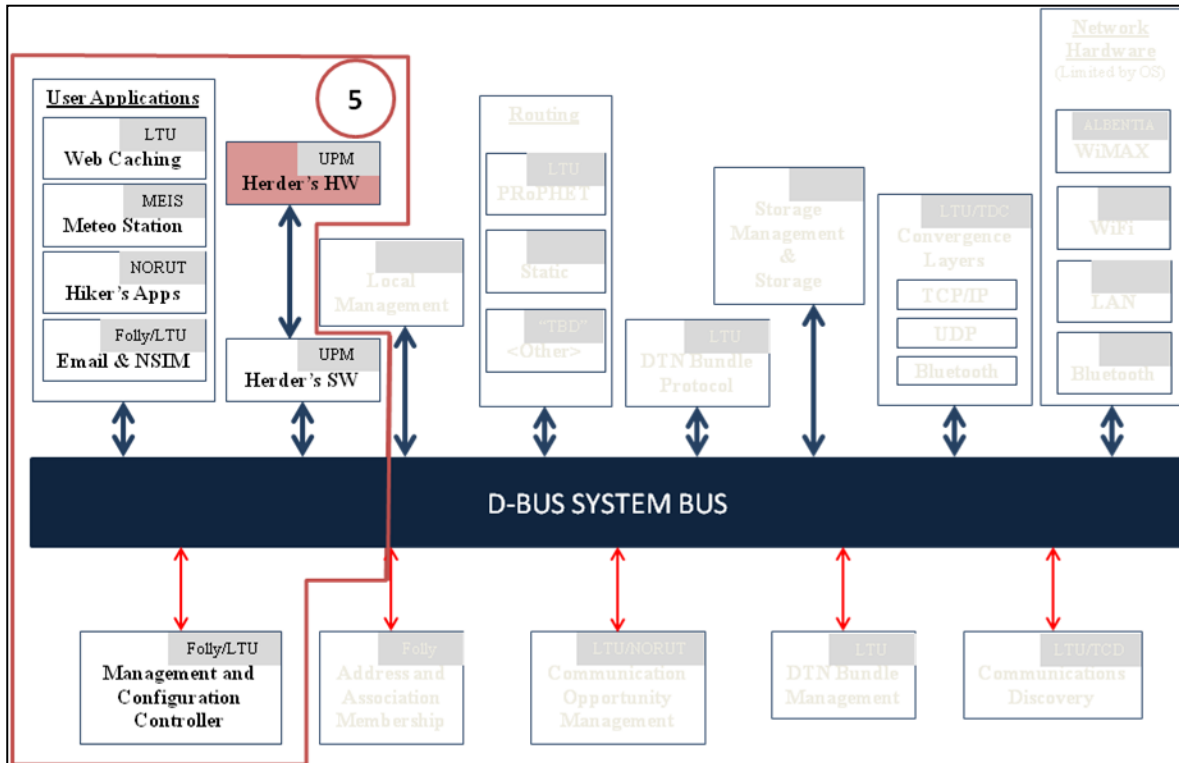


Figure 13: Definition of the subsystem 5 – System Configuration – in the general diagram.

5.7 SUBSYSTEM 6 – ROUTING

As depicted in Figure 14 [4], subsystem 6, named Routing, is comprised of six modules: User Applications, Address and Association Membership, Herder’s Software, Herder’s Hardware, Local Management and Routing. The aim of this subsystem is to test the capabilities of routing, address and new member association.

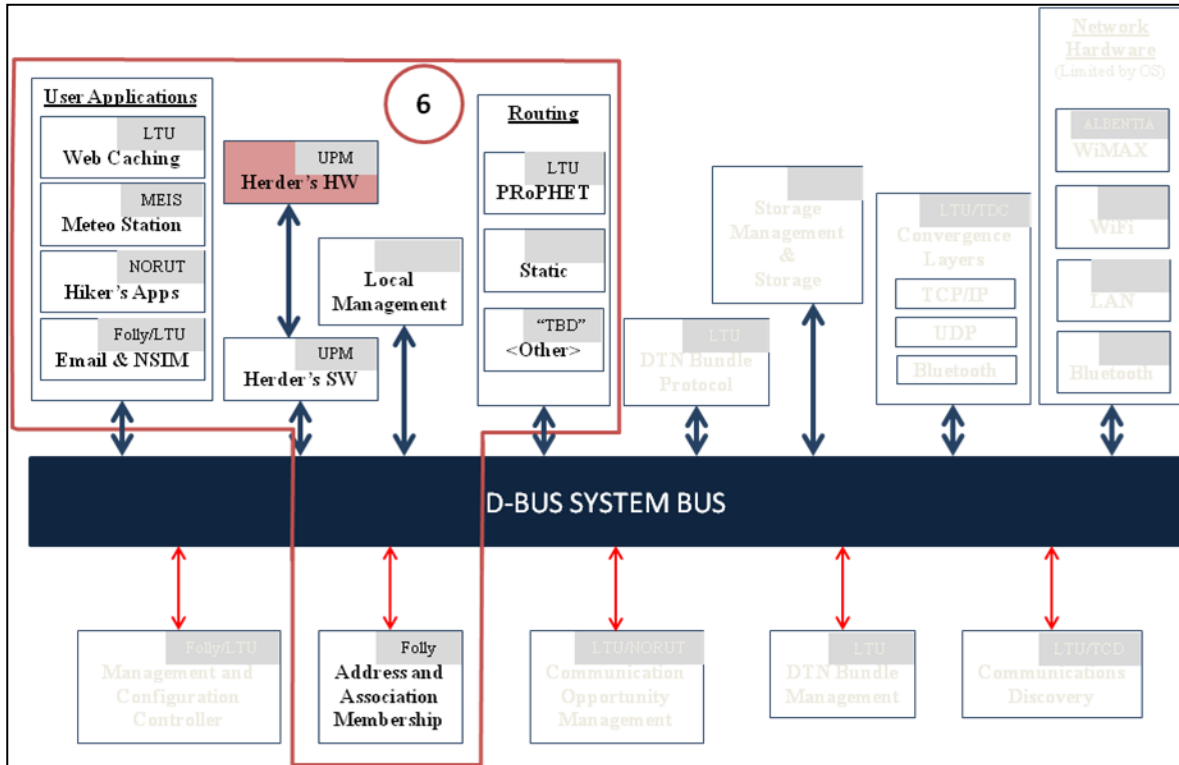


Figure 14: Definition of the subsystem 6 – Routing – in the general diagram.

6. SUBSYSTEM TEST PLANNING

This chapter intends to provide the planning of the tests to all modules, as an initial approach to the next phase of designing the tests themselves. This test plan derives from the chosen test methodology – requirements-based testing, exposed in section 4 -, from a black-box perspective.

In order to test the integration of the system’s modules, this chapter describes, beyond the goal of each module, the initial conditions, as well as some constraints, so tests can be properly made and results can be as reliable and truth as possible.

6.1 LINK AND PHYSICAL SWIM LANES

6.1.1 WI-FI

6.1.1.1 GOAL

Test the reliability and performance of data’s transmission through Wi-Fi technology between two or more nodes.

6.1.2 WIMAX

6.1.2.1 GOAL

Test the reliability and performance of transmission data through WiMax technology between two or more nodes.

6.1.3 BLUETOOTH

6.1.3.1 GOAL

Test the reliability and performance of transmission data through Bluetooth technology between two or more nodes.

6.2 TRANSPORT SWIM LANE

6.2.1 PROPHET

6.2.1.1 GOAL

Test the capabilities and performance of this routing protocol between two or more nodes.

Right now there are two implementation of the PROPHET protocol. One is the stand alone PROPHET implementation from the SNC project and the second one is done within the DTN2 reference implementation. Which implementation will actually be used for further development and testing is still an open question.

6.2.1.2 INITIALIZATION

Tests should start with default values of the PROPHET parameters (alpha 0.75, beta 0.25, gamma 0.98). Later on, the parameters can be tuned up to achieve optimal routing performance on node mobility's case.

All the routing information (such as delivery probability tables on nodes) should be cleared out before every test. All the old log files should be removed as well before every test.

6.2.1.3 ASSUMPTIONS AND RESTRICTIONS

In case of limited time tests, with low node's mobility, it will be prepared and applied an optimized probabilities table to eliminate the needed routing "set up" time.

6.2.1.4 INPUT

Main input for PROPHET testing will be generated network traffic, number of nodes, node mobility and encountering time. In case of a heavy traffic load, node's storage capacity can be used as well.

6.2.1.5 OUTPUT

Gathered traces of the encounters, encountered time, bundle routes, power on/off time and updates of probability tables.

6.2.1.6 TEST CASE PASS/FAIL CRITERIA

After every test, an in-depth analysis should be performed in order to extract the actual routes of the bundles. Main success criteria is whether the protocol can choose, in every node, an optimal route in every "decision steps" or not, according to the gathered information routing provided by each node.

6.2.2 DTN BUNDLE PROTOCOL

6.2.2.1 GOAL

Test the DTN bundle protocol's capabilities and performance between two or more nodes and also the performance when interacting with other protocols.

6.3 APPLICATION SWIM LANE

6.3.1 EMAIL

6.3.1.1 GOAL

Test the capability of sending and receiving email messages with text and attachments. The tests should be done between DTN email servers only and a DTN email server with a LI email server.

6.3.2 NOT SO INSTANT MESSENGER

6.3.2.1 GOAL

Test the capabilities of sending and receiving messages with text or attachments. The tests should be done within the DTN network only.

6.3.2.2 INITIALIZATION

All the old sent and received messages will be removed from all the nodes (including log files).

6.3.2.3 INPUT

Main input will be the messages sent out by the DTN users.

6.3.2.4 OUTPUT

Traces of sent and delivered messages during the test time.

6.3.2.5 TEST CASE PASS/FAIL CRITERIA

To pass this test case, all the messages should be delivered, except those dropped by the network itself, due to an expired bundle time.

6.3.3 WEB CACHING

6.3.3.1 USER PULLED REGULAR

6.3.3.1.1 GOAL

Test if a user can request some data or web pages. These data should be delivered regularly.

6.3.3.1.2 INITIALIZATION

Clients need to be connected to a DTN. The testing equipment should be up and running with all the necessary applications and scripts. A provider/client on LI needs to be connected to a DTN gateway.

6.3.3.1.3 ASSUMPTIONS AND RESTRICTIONS

Requests are made from hotspots or end user clients with DTN and cache applications. It is not certain that the client itself has this software. In that case, the client needs to configure the web browser to use the proxy settings, so it will use the hotspot instead. For now, the user pulled regular is the application that is in testing and should work. The other test cases for web caching will be developed for future tests.

6.3.3.1.4 INPUT

Site/search requests from clients using ad hoc TCP/IP mode and data from providers using TCP/IP.

6.3.3.1.5 OUTPUT

Delivered web data to client.

6.3.3.1.6 TEST CASE PASS/FAIL CRITERIA

Pass: In a controlled test environment, the requests should be processed and tried to be delivered.

Fail: Data is not processed.

6.3.3.2 USER PULLED EVENT DRIVER

6.3.3.2.1 GOAL

Test if a user can set up a cache in advance to provide data.

6.3.3.2.2 *INITIALIZATION*

Clients need to be connected to a DTN. The testing equipment should be up and running with all the necessary applications and scripts. A provider/client on LI needs to be connected to a DTN gateway.

6.3.3.2.3 *ASSUMPTIONS AND RESTRICTIONS*

Requests are made from hotspots or end user client with DTN and cache applications. It is not certain that client itself has this software. In that case, the client needs to configure the web browser to use the proxy settings so it will use the hotspot instead. For now, the user pulled regular is the one that is in testing and should work. The other use cases will be worked on for future tests.

6.3.3.2.4 *INPUT*

Site/search requests from clients using ad hoc TCP/IP mode and data from providers using TCP/IP.

6.3.3.2.5 *OUTPUT*

Delivered web data to client.

6.3.3.2.6 *TEST CASE PASS/FAIL CRITERIA*

Pass: In a controlled test environment, the requests should be processed and tried to be delivered.

Fail: Data is not processed.

6.3.3.3 *PROVIDER PUSHED REGULAR*

6.3.3.3.1 *GOAL*

Test if the provider can regularly deliver important information into the DTN network (e.g. information for tourists or governmental information).

6.3.3.3.2 *INITIALIZATION*

Clients need to be connected to a DTN. The testing equipment should be up and running with all the necessary applications and scripts. A provider/client on LI needs to be connected to a DTN gateway.

6.3.3.3.3 *ASSUMPTIONS AND RESTRICTIONS*

Requests are made from hotspots or end user client with DTN and cache applications. It is not certain that client itself has this software. In that case, client needs to configure the web browser to use the proxy setting so it will use the hotspot instead. For now, the user pulled regular is the one that is in testing and should work. The other use cases will be worked on for future tests.

6.3.3.3.4 INPUT

Site/search requests from clients using ad hoc TCP/IP mode and data from providers using TCP/IP.

6.3.3.3.5 OUTPUT

Delivered web data to client.

6.3.3.3.6 TEST CASE PASS/FAIL CRITERIA

Pass: In a controlled test environment, the requests should be processed and tried to be delivered.

Fail: Data is not processed.

6.3.3.4 PROVIDER PUSHED EVENT DRIVEN**6.3.3.4.1 GOAL**

Test if the provider can send data to cache in advance (e.g. educational contents).

6.3.3.4.2 INITIALIZATION

Clients need to be connected to a DTN. The testing equipment should be up and running with all the necessary applications and scripts. A provider/client on LI needs to be connected to a DTN gateway.

6.3.3.4.3 ASSUMPTIONS AND RESTRICTIONS

Requests are made from hotspots or end user clients with DTN and cache applications. It is not certain that the client itself has this software. In that case, the client needs to configure the web browser to use the proxy settings, so it will use the hotspot instead. For now, the user pulled regular is the one that is in testing and should work. The other use cases will be worked on for future tests.

6.3.3.4.4 INPUT

Site/search requests from clients using ad hoc TCP/IP mode and data from providers using TCP/IP.

6.3.3.4.5 OUTPUT

Delivered web data to client.

6.3.3.4.6 TEST CASE PASS/FAIL CRITERIA

Pass: In a controlled test environment, the requests should be processed and tried to be delivered.

Fail: Data is not processed.

6.3.3.5 AD HOC SITE REQUEST

6.3.3.5.1 GOAL

Test if a user can request a site that is not normally cached.

6.3.3.5.2 INITIALIZATION

Client need to be connected to a DTN. The testing equipment should be up and running with all the necessary applications and scripts. A provider/client on LI needs to be connected to a DTN gateway.

6.3.3.5.3 ASSUMPTIONS AND RESTRICTIONS

The request is made from hotspots or end user client with DTN and cache applications. It is not certain that client itself has this software. In that case, the client needs to configure the web browser to use the proxy settings, so it will use the hotspot instead. For now, the user pulled regular is the one that is in testing and should work. The other use cases will be worked on for future tests.

6.3.3.5.4 INPUT

Site/search requests from clients using ad hoc TCP/IP mode and data from providers using TCP/IP.

6.3.3.5.5 OUTPUT

Delivered web data to client.

6.3.3.5.6 TEST CASE PASS/FAIL CRITERIA

Pass: In a controlled test environment, the requests should be processed and tried to be delivered.

Fail: Data is not processed.

6.3.3.6 AD HOC SEARCH REQUEST

6.3.3.6.1 GOAL

Test if a user can make a search request through a search box.

6.3.3.6.2 INITIALIZATION

Client need to be connected to a DTN. The testing equipment should be up and running with all the necessary applications and scripts. A provider/client on LI needs to be connected to a DTN gateway.

6.3.3.6.3 ASSUMPTIONS AND RESTRICTIONS

Requests are made from hotspots or end user clients with DTN and cache applications. It is not certain that client itself has this software. In that case, the client needs to configure the web browser to use the proxy settings, so it will use the hotspot instead. For now, the user pulled regular is the one that is in testing and should work. The other use cases will be worked on for future tests.

6.3.3.6.4 INPUT

Site/search requests from clients using ad hoc TCP/IP mode and data from providers using TCP/IP.

6.3.3.6.5 OUTPUT

Delivered web data to client.

6.3.3.6.6 TEST CASE PASS/FAIL CRITERIA

Pass: In a controlled test environment, the requests should be processed and tried to be delivered.

Fail: Data is not processed.

6.3.3.7 METEOROLOGICAL STATION

6.3.3.7.1 GOAL

Test data's transfer from meteorological station (temperature, relative humidity, wind speed and direction, pressure, solar radiation and precipitation) through the DTN to LI. This must be tested in two different ways: In a periodical time schedule and whenever the transmission is requested (as faster as possible).

6.3.3.7.2 INITIALIZATION

Several initialization files will be prepared for the data transfer: init file for meteorological station, defining the constants for the sensors; init file for meteorological station defining the processing of the measured data, init file for meteorological station defining the data transmitting and init file for the DTN protocol defining the destination of data.

6.3.3.7.3 ASSUMPTIONS AND RESTRICTIONS

All hardware should be able to work within a temperature range from -20 to +80 degrees Celsius.

6.3.3.7.4 INPUT

Data from meteorological sensors:

- Air temperature
- Relative humidity
- Wind speed and direction
- Pressure
- Global solar radiation
- Precipitation

6.3.3.7.5 OUTPUT

Data files in ASCII format (see D3.1 Functional Specification (Initial), document:

N4C-WP3-2-fs-initial-04.doc).

6.3.3.7.6 TEST CASE PASS/FAIL CRITERIA

Pass: 75% of data successfully transferred to the destination.

Fail: A percentage of successfully transferred data below 75%.

6.3.4 HIKER'S APPLICATIONS

6.3.4.1 POINTS OF INTEREST (POI)

6.3.4.1.1 GOAL

Test the download of interest points. This test case tests the application in terms of extraction of GPS maps with location of interest points, from the web cache/internet. The location of nearest medical services (medical, physical or heart starter) can be one type of Point Of Interest (POI). (It is possible a pro-actively implementation of this requirement.)

6.3.4.1.2 INITIALIZATION

The client needs to be connected to a DTN-node with web cache of POIs for the area. The testing equipment should be up and running with all the necessary applications and scripts. GPS must be turned on for a minute or so to get a fixed position.

6.3.4.1.3 ASSUMPTIONS AND RESTRICTIONS

When tests begin, the GPS receiver has already found its initial position.

6.3.4.1.4 INPUT

GPS Position.

6.3.4.1.5 OUTPUT

POI database updated.

6.3.4.1.6 TEST CASE PASS/FAIL CRITERIA

Pass: POI database updated.

Fail: POI database not updated.

6.3.4.2 SEND MESSAGE WITH OWN LOCATION

6.3.4.2.1 GOAL

Test the process of sending a message with one's own location to LI.

6.3.4.2.2 INITIALIZATION

The client needs to be connected to a DTN-node. The testing equipment should be up and running with all the necessary applications and scripts. GPS must be turned on for a minute or so to get a fixed position.

6.3.4.2.3 ASSUMPTIONS AND RESTRICTIONS

When tests begin, the GPS receiver has already found its initial position.

6.3.4.2.4 INPUT

GPS Position.

6.3.4.2.5 OUTPUT

Email with GPS Position.

6.3.4.2.6 TEST CASE PASS/FAIL CRITERIA

Pass: Email received at destination.

Fail: No email received at destination.

6.3.4.3 MAPS FOR OWN LOCATION**6.3.4.3.1 GOAL**

Test the process of downloading maps of the area around one's own location.

6.3.4.3.2 INITIALIZATION

The client needs to be connected to a DTN-node with web cache of maps for the area. The testing equipment should be up and running with all the necessary applications and scripts. GPS must be turned on for a minute or so to get a fixed position.

6.3.4.3.3 ASSUMPTIONS AND RESTRICTIONS

When tests begin, the GPS receiver has already found its initial position.

6.3.4.3.4 INPUT

GPS Position.

6.3.4.3.5 OUTPUT

Map cache updated.

6.3.4.3.6 TEST CASE PASS/FAIL CRITERIA

Pass: Map cache updated.

Fail: Map cache not updated.

6.3.4.4 GEOBLOG AND PHOTOBLOG**6.3.4.4.1 GOAL**

Test the automatic actualization of geoblog and photo blog when there is a connection opportunity.

6.3.4.4.2 *INITIALIZATION*

Client need to be connected to a DTN-node. The testing equipment should be up and running with all the necessary applications and scripts. GPS must be turned on for a minute or so to get a fixed position.

6.3.4.4.3 *ASSUMPTIONS AND RESTRICTIONS*

When tests begin, the GPS receiver has already found its initial position.

6.3.4.4.4 *INPUT*

GPS Position.

6.3.4.4.5 *OUTPUT*

Blog with GPS Position or photo with GPS Position.

6.3.4.4.6 *TEST CASE PASS/FAIL CRITERIA*

Pass: Blog with GPS Position, or photo with GPS Position received at destination.

Fail: No blog with GPS Position, or no photo with GPS Position received at destination.

6.3.5 HERDER APPLICATION AND ANIMAL MIGRATION MONITORING

6.3.5.1 *GOAL*

Test the system of the herder application and animal migration monitoring, checking the quality and reliability of the connection between the equipment on animal and the receiver's station. This must be tested in two different ways: In a periodical time schedule, issued by the animal's equipment, and whenever the transmission is requested.

6.3.6 MANAGEMENT AND CONFIGURATION CONTROLLER

6.3.6.1 GOAL

Test management applications for configuration, monitoring and analysis of the DTN user applications and infrastructure. It will be tested the operability and reliability of applications to:

- determine when DTN capability is available in a node;
- determine the modes of communication currently available (DTN bundle protocol, LTP transport, direct connection to LI, etc);
- open, manage and close communication channels using DTN protocols, providing appropriated control for both end user applications and proxy servers;
- send and receive messages over the DTN infrastructure;
- receive notifications when the modes of communication alter due to movement or reconfiguration of the node;
- control and use security mechanisms to allow authentication and encryption for messages;
- control and monitor the addressing of the node;
- control and monitor DTN routing mechanisms;
- control and monitor message (packet, bundle, etc) storage and forwarding.

7. CONCLUSIONS

From the methodologies exposed in section 3.2, we came to the conclusion that the best suitable integration methodology is IPC, more concretely, the D-Bus integration platform, due to the reasons already explained in section 5, and also because this method is supported in the current most usual operating systems, like Windows, Linux and Apple S.O. X.

Having this methodology in mind, next summer's integration tests can be performed according to the specifications made under sections 5.2-5.7, using the initial conditions and restrictions mentioned in chapter 6. These tests have, however, to be completely defined and designed in a near future, in order to be available and completely specified before the summer tests.

8. REFERENCES

- [1] *D-Bus Tutorial*. (2009). Retrieved April 15, 2009, from <http://www.freedesktop.org/wiki/Software/dbus>
- [2] Inc., B. R. (2003). *Requirements Based Testing*. Retrieved April 15, 2009, from <http://benderbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf>
- [3] *Integration Test Plan*. (s.d.). Retrieved April 15, 2009, from http://www.cc.gatech.edu/classes/cs8113h_96_spring/process/Integration_Test_Plan.html
- [4] (2009). *M7.1 – N4C Subsystem Integration Model and Tests*.
- [5] (2009). *M7.2 – N4C Preparation of Specification of the Integration Platform and Guidelines for Implementation*.
- [6] Tsai, W.-T. (2003). *Integration Testing*. Retrieved April 15, 2009, from <http://asusrl.eas.asu.edu/cse494/content/integration/Integration%20Testing.PDF>
- [7] Williams, L. (2006). *Testing Overview and Black-Box Testing Techniques*. Retrieved April 15, 2009, from <http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf>