Project no: 269317

nSHIELD

new embedded Systems arcHItecturE for multi-Layer Dependable solutions

Instrument type: Collaborative Project, JTI-CP-ARTEMIS

Priority name: Embedded Systems

D7.3: Dependable Avionic System demonstrator - integration and validation plan

Due date of deliverable: M22 –2013.06.30

Actual submission date: M22 –2013.06.30

Start date of project: 01/09/2011 Duration: 36 months

Organisation name of lead contractor for this deliverable:

Selex-ES, SES

Revision [Issue 1]

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2012)		
Dissemination Level		
PU	Public	
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)	**X**

Document Authors and Approvals

Authors		Date	Signature
Name	**Company**		
Massimo Traversone	Selex ES		
Marina Guzzetti	Selex ES		
Antonio Di Marzo	SESM		
Antonio Bruscino	SESM		
Tor O Steine	Alfatroll AS		
Kresimir Dabcevic	UNIGE		
Lucio Marcenaro	UNIGE		
Iñaki Eguia	Tecnalia		
Andrea Morgagni	Selex ES		
Roberto Binaghi	Selex ES		
Andrea Fiaschetti	UNIROMA		
Balazs Berkes	S-LAB		
Kiriakos Georgouleas	HAI		
Nikos Pappas	HAI		
Reviewed by			
Name	**Company**		
Approved by			
Name	**Company**		

Applicable Documents		
ID	**Document**	**Description**
[01]	TA	nSHIELD Technical Annex

Modification History		
Issue	**Date**	**Description**
Draft 01	30.04.2013	First version of TOC
Draft 02	10.05.2013	Added list of nSHIELD prototypes outcome of the Stockolm meeting; added Annex A for ICDs; updated descriptions; editorial work
Draft 03	16.05.2013	Added proposed template for ICDs; added SPD features subsections; updated descriptions.
Draft 04	16.06.2013	Added draft description of the avionic scenario
Draft 05	27.06.2013	Added description of nS-ESD-GW abd IQ Engine
Draft 06	22.06.2013	Added description of OMBRA, Multi-metrics, Surface metrics, Middleware Intrusion detection system and Protection profile
Draft 07	30.06.2013	Added description of the Dependable Avionic scenario, Semantic model and SDP features, OSGI middleware

Contents

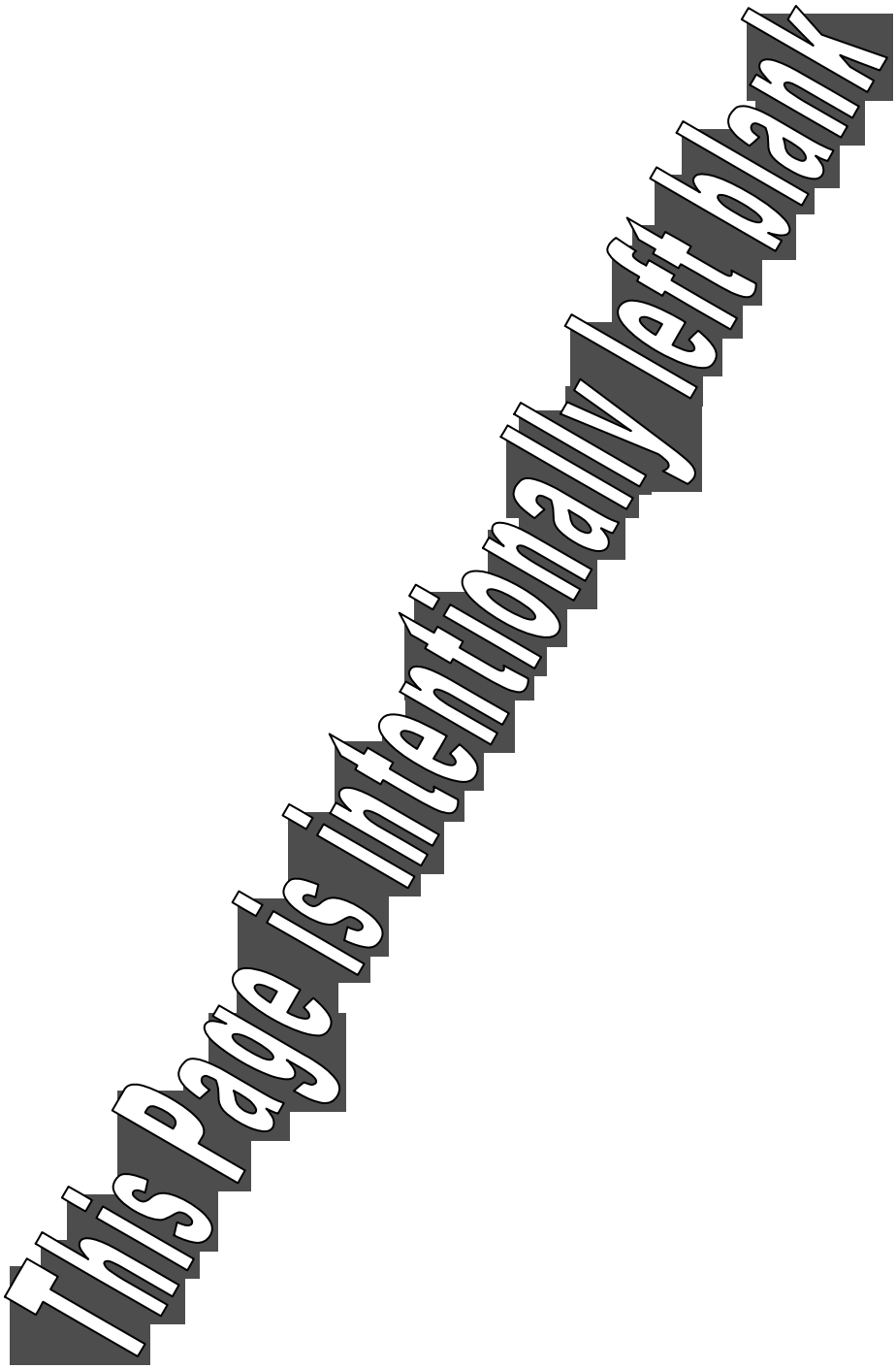
Figures

Tables

No table entries found.

Glossary

Please refer to the Glossary document, which is common for all the deliverables in nSHIELD.


This Page is Intentionally left blank

1 Executive Summary

The purpose of this document is to present the plan and methodologies driving the integration, the validation and verification activities for the nSHIELD Dependable Avionic Scenario. The document is structured as follows:

• Chapter 2 – provides a brief introduction on Dependable Avionic System demonstrator structure

• Chapter 3 – presents the SHIELD taxonomy, and a table with all nSHIELD prototypes

• Chapter 4 – presents detailed description of Dependable Avionic System Demonstrator scenario and the nSHIELD solution proposed

• Chapter 5 – presents the reference architecture for Dependable Avionic System Demonstrator

• Chapter 6 – presents the prototypes involved in the demonstrator scenario and their SPD characteristics

• Chapter 7 – presents the Dependable Avionic System Demonstrator functionalities

• Chapter 8 – presents the Avionic System Demonstrator integration activities

• Chapter 9 – presents the Avionic System Demonstrator integration V&V approach.

• Chapter 10 – draws the conclusions

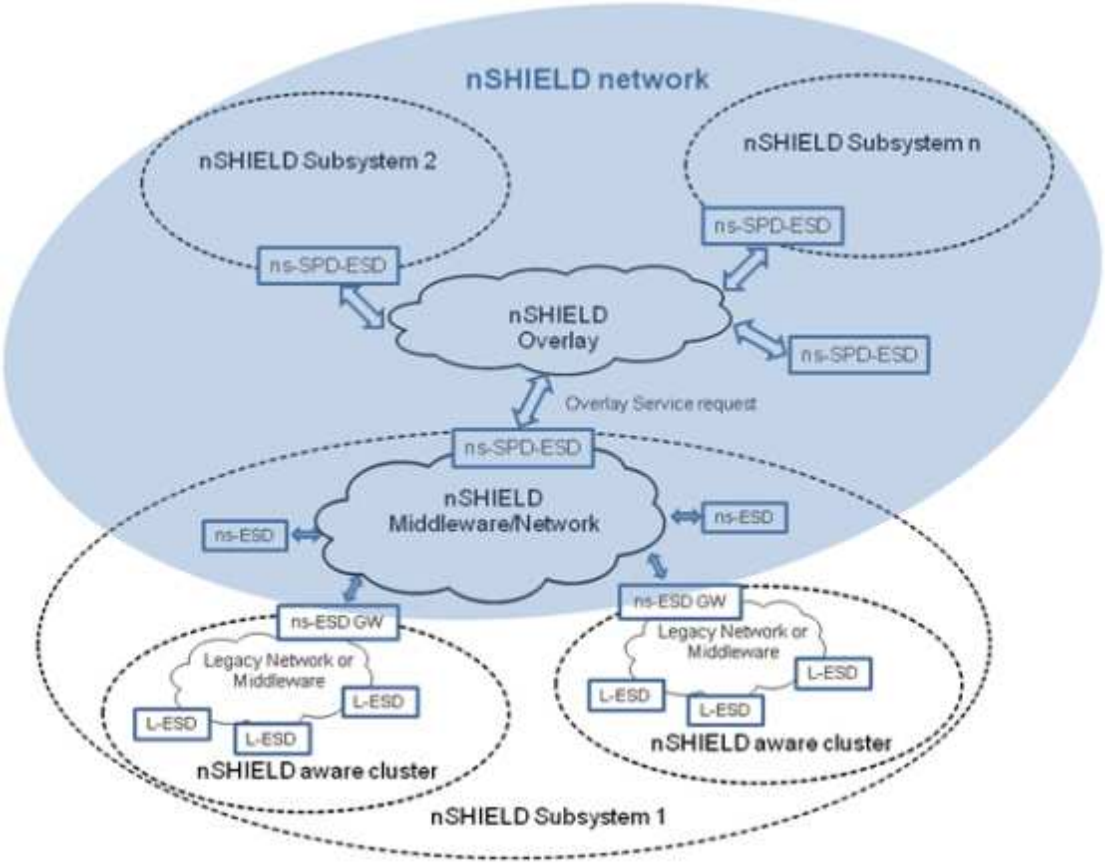
• Appendix A – presents a detailed description of some prototypes interfaces

2 Introduction

2.1 Avionic Dependable Demonstrator structure

This section identifies how the nSHIELD Dependable Avionic System scenario is structured

- SHIELD framework will be employed to design an innovative Avionic Dependable Architecture
- Aspect such as Dependability and Composability will be encompassed into the demonstrator



.

Figure 2-1 High Level Avionic Dependable Demonstrator Structure

Mainly the Dependable Avionic scenario can be seen as System of Systems (SOS): a set of heterogeneous systems logically or physically connected that cooperate for the execution of one or more tasks.

Typical examples of applications of system of systems are the “surveillance systems” for several different application like search & rescue or security.

The term surveillance is a set of techniques, devices and methodologies to detect, prevent and recognize “behavior” in order to prevent/investigate all the possible behavior.

The scenario of the Dependable Avionic system for surveillance is taken as a reference in the demonstrator as particularly suited to highlight the potential of the methodology of the nSHIELD. Typically

a system for surveillance is composed of a large number of heterogeneous systems (radar, sonar, …) including aircraft of many different grades and sizes (typically unmanned : UAV).

Each operational level must be able to communicate with its own network operating according to a certain protocol and with certain benefits. For the security of the surveillance is vital that communication between different levels must be guaranteed with an adequate level of security and dependability, worth the loss of the main functions of the system.

Given the UAV as one of the N subsystems of the security of the surveillance, the goal of this demonstrator is to develop a number of hardware and software modules to be integrated appropriately into the surveillance subsystem.

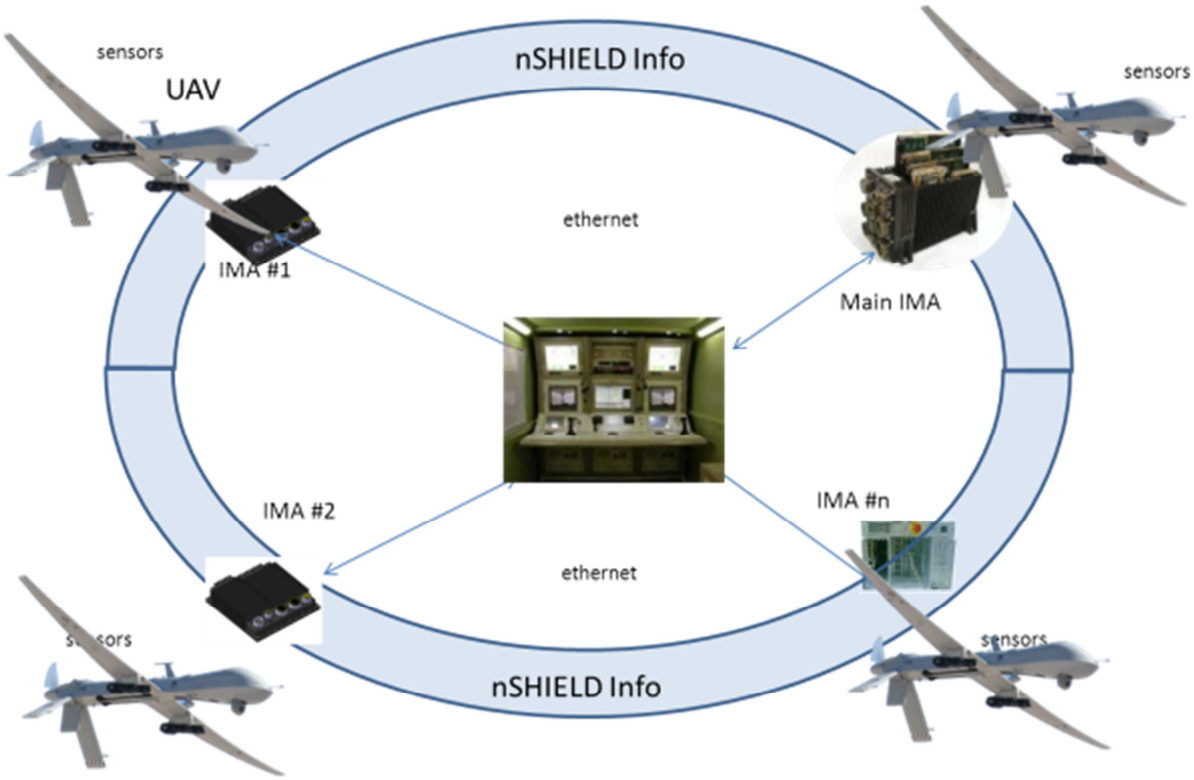


Figure 2-2 Dependable Avionic Scenario

3 Terms and Definitions

3.1 nSHIELD prototypes

A number of individual prototypes have been identified in the nSHIELD framework. They are enumerated in the following table and associated to an Id code.

Id	Name	Author
00	Elliptic Curve Cryptography	UNIGE
01	Lightweight Cyphering	TUC
02	Key Exchange Protocol	TUC
03	Hypervisor	SICS
04	Secure Boot	T2D
05	Smart Power Unit	AS
06	Smart Card	TUC
07	Facial Recognition	ETH
08	GPU Hase	TUC
09	SPD-driven Smart Transmission Layer	SES/UNIGE
10	Anonymity	TUC
11	Automatic Access Control	TUC
12	DDoS Attack Mitigation	ATHENA
13	Recognizing DoS	ATHENA
14	Cellular Automata	UNIUD
15	Intrusion Detection System	MGEP
16	Reputation-Based Secure Routing	TUC/HAI
17	Access Control Smart Grid	TECNALIA
18	Policy Definition	ASTS/SES/SESM
19	Policy Based Management Framework	TUC/HAI
20	Control Algorithms	UNIROMA
21	Gateway	SESM
22	Middleware Intrusion Detection System	S-LAB

23	Link Layer Security	INDRA
24	Network Layer Security	TUC
25	OSGI Middleware	UNIROMA1
26	Semantic Model	UNIROMA1
27	Multimetrics	TECNALIA
28	Attack Surface Metrics	SES
29	Adaptation of Legacy System	ATHENA
30	Reliable Avionic	ALFATROLL
31	Protection Profile	SES
32	Secure Discovery	UNIROMA1
33	Secure Agent	UNIROMA1
34	Audio Surveillance System	ISD
35	BeagleBoard-Xm	SICS
36	OMNIA-IMA	SES

4 nSHIELD Dependable Avionic scenario

In this section the dependable avionic scenario, at the heart of the chosen demonstrator, will be described. The Unmanned Aerial Vehicle (UAV) is a system composed by an aircraft controlled by a pilot located on the ground, through a remote control unit. The avionic equipment of the UAV has been designed in adherence to the IMA (Integrated Modular Avionic) concept:

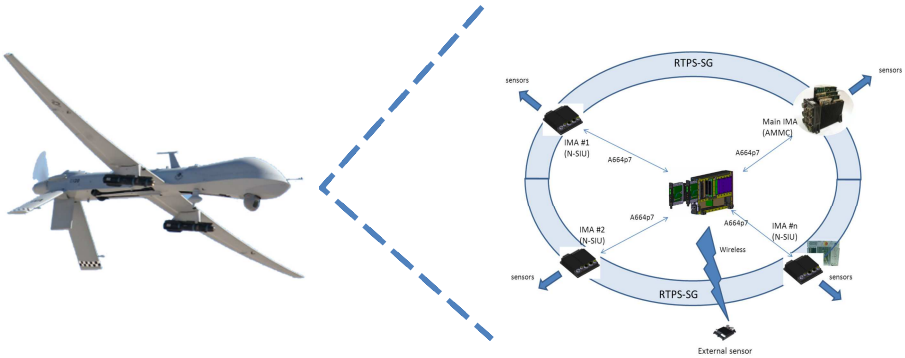


Figure 4-1 UAV Avionic Equipment

Two type of faults will be tackled in the context of the avionic demonstrator; and the SHIELD countermeasure mechanisms will be shown.

With the aim to demonstrate the fault detection within the UAV architecture, an initial configuration of the scenario is made up of a single UAV and the remote control unit.

During a normal operational status, a hardware fault in the IMA will be injected. Such an event can be overcome with the usage of a spare unit. Thanks to the adoption of the NSHIELD methodology, already embedded into the IMA components, the fault will be identified, isolated and recovered. In this way mission success won't be jeopardized at all.

Regarding the second type of fault, in the context of system of systems an error condition will be identified and isolated with the aid of the system composability. In particular, the demonstrator boundaries will be extended in a way to includes a second UAV (or a different suitable node) and other nSHIELD actors. Such a new system configuration will be executed at runtime by means of the joint action of the nSHIELD Middleware and Overlay.

The second fault is experienced on the UAV positioning system, in particular a data discrepancy among onboard sensor readings is detected. Thanks to the intervention of the nSHIELD Middleware and Overlay, a fault recovery mechanism is triggered. In order to overcome this system impairment, a fixing on the faulty UAV is performed using sensor readings taken from the second UAV in close proximity of the faulty one.

With the intervention of the fully functional sensor, data reconciliation is then performed through the new nSHIELD node. The communication between these two nodes is carried on thanks to the integration of the nS-ESD-GW and SPD-driven Smart Transmission Layer board that allow the exchanging of information through the nSHIELD middleware, the orchestrator of all these complex activities. In this way the faulty sensor is isolated.

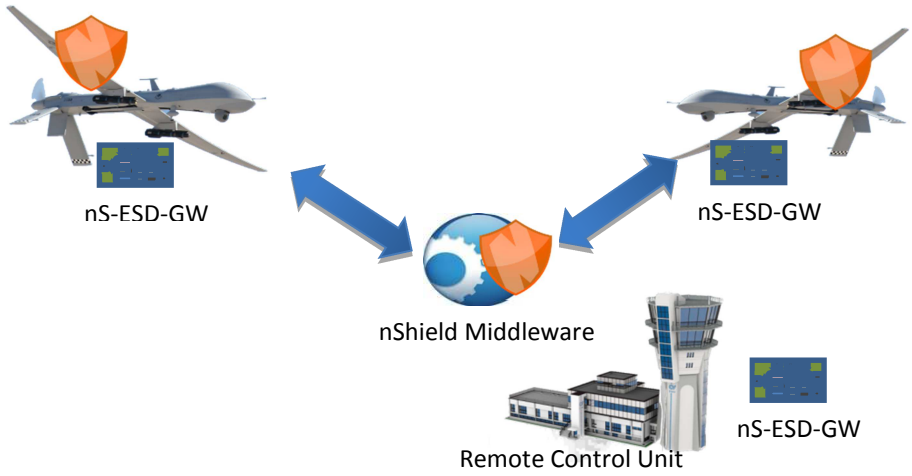


Figure 4-2 Avionic scenario outline

Unlike a conventional standard avionic FDI, nSHIELD methodology can overcome faults both at system and system of systems level as well. In this way, dependability and composability concepts can be shown in a very effective manner.

A configurable gateway capable of easing the integration of legacy embedded systems into a nSHIELD network is used. The gateway (dubbed as nS-ESD-GW) acts as a “proxy” that connects legacy embedded systems to the nSHIELD network and middleware.

The detail of the interfaces exposed by components is reported in Appendix A for integration purposes.

5 Dependable Avionic System demonstrator reference architecture

The whole demonstrator is a rather complex architecture. Indeed, due to scenario complexity, the demonstrator will be delivered as a laboratory exercise. Nonetheless, the opportunity to deploy a mock-up is being evaluated as well. The main actors of the demonstrator are: OMNIA, gateway (nS-ESD-GW) and nSHIELD Middleware. However we do envision the integration in the aforementioned scenario of IQ_Engine and SPD-driven Smart Transmission Layer.

Due to budget, time and complexity constraints, a fully functional system isn't implemented at all. For this reason a bench demonstrator is employed. Albeit this simplification, key concepts are still shown in full (composability and dependability) and in a very effective way.

Hereafter a brief description of every component is given:

- A stripped down version of the UAV1 is represented by a rack unit equipped with: a simulated set of sensors (system camera, GPS and inertial sensors), AMMC (Aircraft & Mission Management Computer) on which runs IQ_Engine, supporting components which ensure communication with other nodes (SDR/Cognitive-capable node), an interface toward the nSHIELD world (nS-ESD-GW);

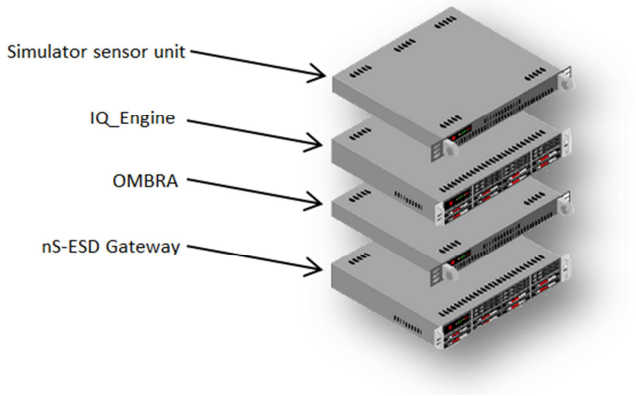


Figure 5-1 UAV demonstrator rack system

- A stripped down version of the UAV2 is identical to UAV1;
- A C.O.T.S. PC equipped with the nSHIELD middleware and the same communication link used by UAVs;
- A stripped down version of the remote control unit implemented thanks to a C.O.T.S. PC equipped with nS-ESD-GW, the same communication link used by UAVs.

As for the prototype listed in para 3.1, the nSHIELD Dependable Avionic Scenario is composed from the following prototypes connected between them :

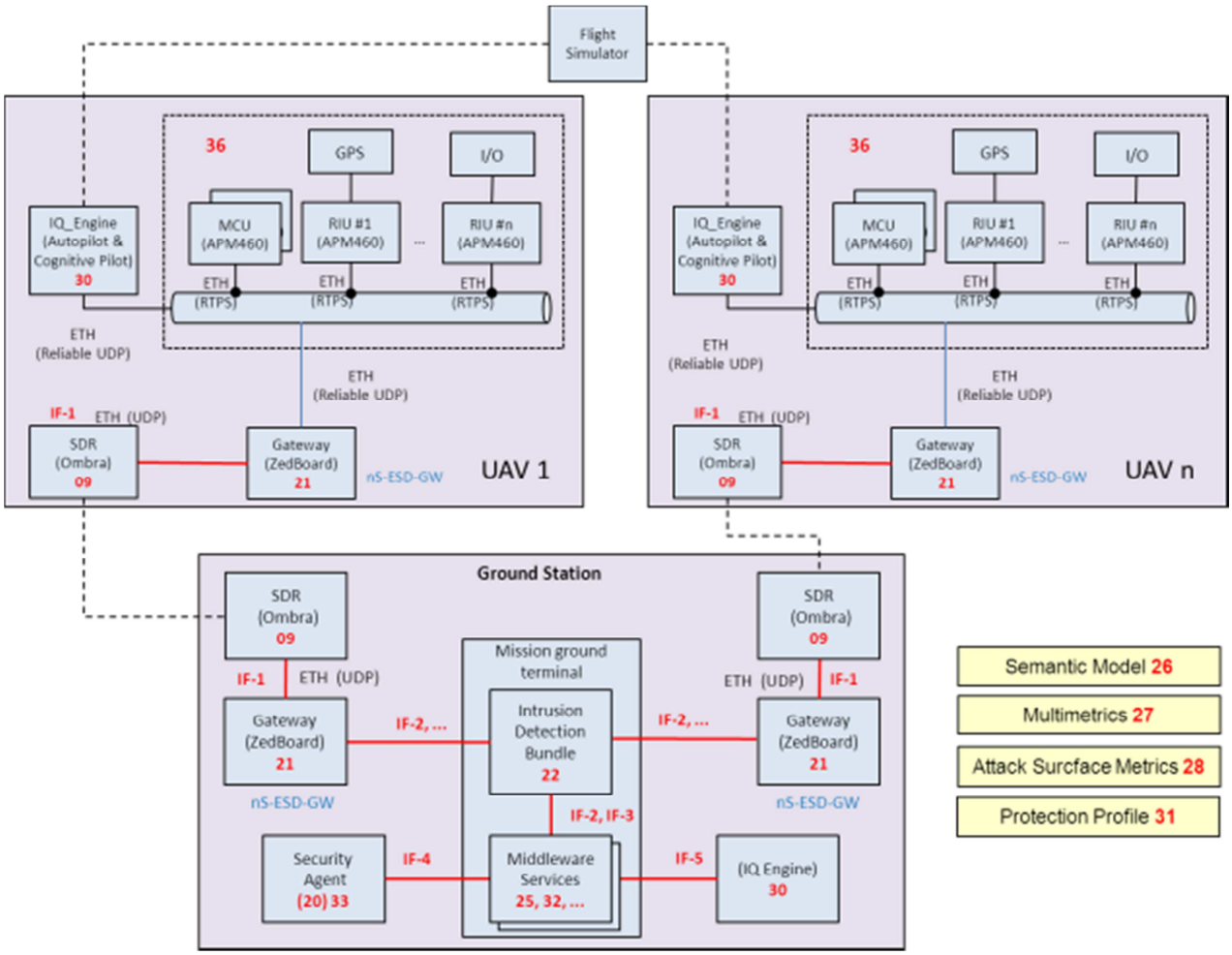


Figure 5-2 Dependable Avionic Scenario Prototypes

The use of these prototypes makes the Dependable avionic scenario nSHIELD-compliant, highlighting the enhancements of SPD solutions obtained with the application of the nSHIELD methodology. In particular, it will be demonstrated:

- Security:
  - The data exchanged between the UAV and the Ground Station will be preserved by interferences. This aspect will be covered thanks to the application of appropriate communication mechanisms in the proposed scenario.
- Privacy:
  - Different access to the UAV for the mission operators will be guaranteed.
- Dependability:
  - The data acquired by on-board sensors are protected against corruption.
  - Automated system recovery.

6 Dependable Avionic System demonstrator technology overview

6.1 OMNIA

OMNIA (Open Mission Network Integrated Architecture) is based on IMA and IMA2G concepts and introduces, at some level, typical nSHIELD dependability aspects, such as interoperability, fault detection, fault management, health monitoring, data integrity…

The idea behind OMNIA is to create an IMA platform composed by a network of several «Computer Units»; these can be HW boards or computers, acting as IMA CU (Central Unit), RIU (Remote Interface Unit) or both. Each unit acting as RIU is connected to the A/C (AirCraft) sensors. All units are “nodes”of a network, being connected by means of an High Speed deterministic serial line.

The OMNIA platform introduces the Middleware software to provide platform level services. The Middleware is implemented on top of the Operating System local to the hardware components. Its purpose is to enable IMA2G typical interoperability, allowing the provision of the same service with the same behavior in such a way that it is independent from the physical location of the requesting applications, i.e. the hardware component that hosts the application, independent from the hardware component type, if applicable, and the Operating System hosted and independent from the location of the requested hardware resources

Platform Services are classified at two levels according to their scope, which can be the overall platform or a single hardware component; Platform Services include in fact Module Services. Platform Services are also classified according to their privilege, whether they support Avionic applications or Platform Management applications (including module management).

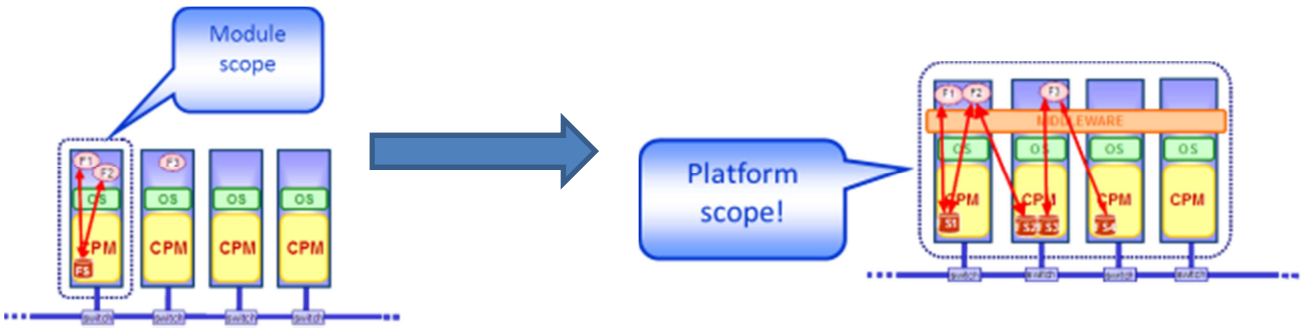


Figure 6-1 Platform Services

More specifically, the “nodes” selected to create the OMNIA network are:

- The NAMMC (New Aircraft & Mission Management Computers)
- The APM460 processor module (stand alone)
- The NSIU (New Sensor Interface Unit)
- More types could be added in future ...

The NAMMC, including HW and Equipment SW, is a SES product, flying onboard several types of aircrafts, able to host the customer Flight Management applications. It is currently completing the certificaton process. It mounts APM460 processor modules and is able to interface with the A/C sensors by means of IO boards (e.g. the DASIO). In the OMNIA platform the NAMMC can act as CU or RIU or both.
The APM460 stand alone can act as CU
The NSIU is a SES computer currently at development stage. It can act as CU or RIU in the OMNIA platform

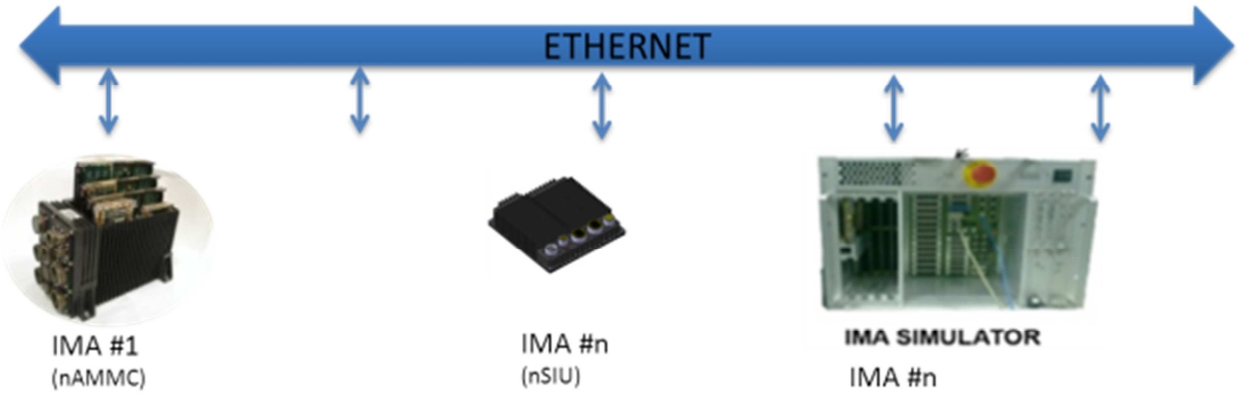


Figure 6-2 OMNIA platform

For the nSHIELD avionic dependable demonstrator, as depicted in the previous figure, it is proposed to configure the OMNIA platform with two or more CU/RIU equipments (NAMMC or NSIU or simulated NAMMC/NSIU) in order to simulate the onboard avionics on a UAV. This configuration can be changed to include more “nodes” if necessary.

A SW middleware based on DDS architecture, with a unique service bus, will be used to “virtualise” the physical connection of the A/C sensors enabling **interoperability** within the OMNIA platform as it will allow the OMNIA platform “nodes” to access the sensor resources independently from the actual physical connection.

Health monitoring and **fault management** within the OMNIA platform are performed at “node” level by means of continuous built in tests. **Integrity** of sensors data will be handled at OMNIA middleware level

For the nSHIELD demonstrator the OMNIA system will be able to provide the main aircraft mission/navigation functionalities with all the relevant check either relating to the data integrity exchanged and to the integrity of the OMNIA system (fail, reconfiguration, …).

6.1.1 OMNIA SPD features

As preliminary definition of each OMNIA features, the following figures represent the different “avionics” layers with the detail of the functionalities.

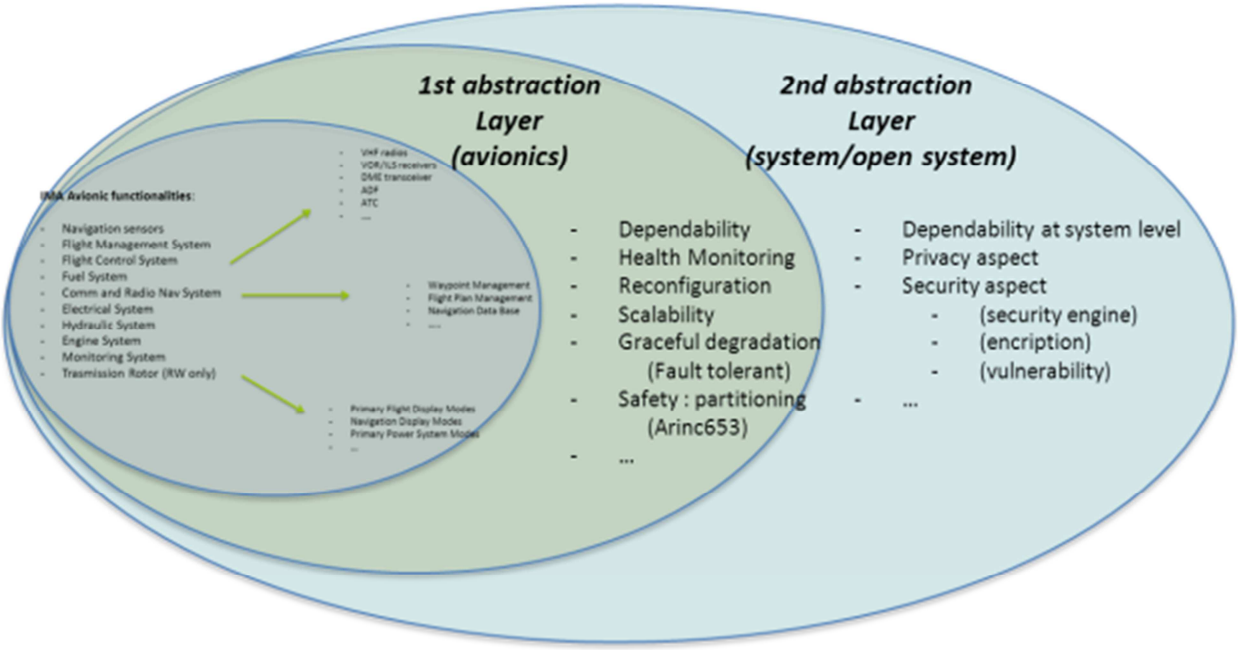


Figure 6-3 Avionics Layers

6.2 Gateway

nS-ESD-GW is a SHIELD framework pivotal component. This component has been introduced to foster the interconnection of nodes and to ease the SHIELD employment. Thus, in the context of the Dependable Avionic application scenario, this component will be employed to ease the integration and interconnection of the OMNIA framework to nSHIELD nodes, middleware and overlay as well. The nS-ESD-GW will exploit the flexibility of the SoC (Zynq) to provide appropriate interfaces towards the OMNIA components. The Zynq consists of an hybrid architecture composed by a dual-core Cortex ARM A9 and a 7-series Xilinx FPGA. The Zynq is an innovative SoC characterized by a powerful ecosystem that greatly simplifies the development process and shrinks the time to market.

The nS-ESD-GW has been designed following a modular approach; this enables the tight partitioning and isolation between internal components involved to implement security, communication and monitoring functions. Furthermore the modularization eases the adaptation process of nS-ESD-GW to the avionic scenario.

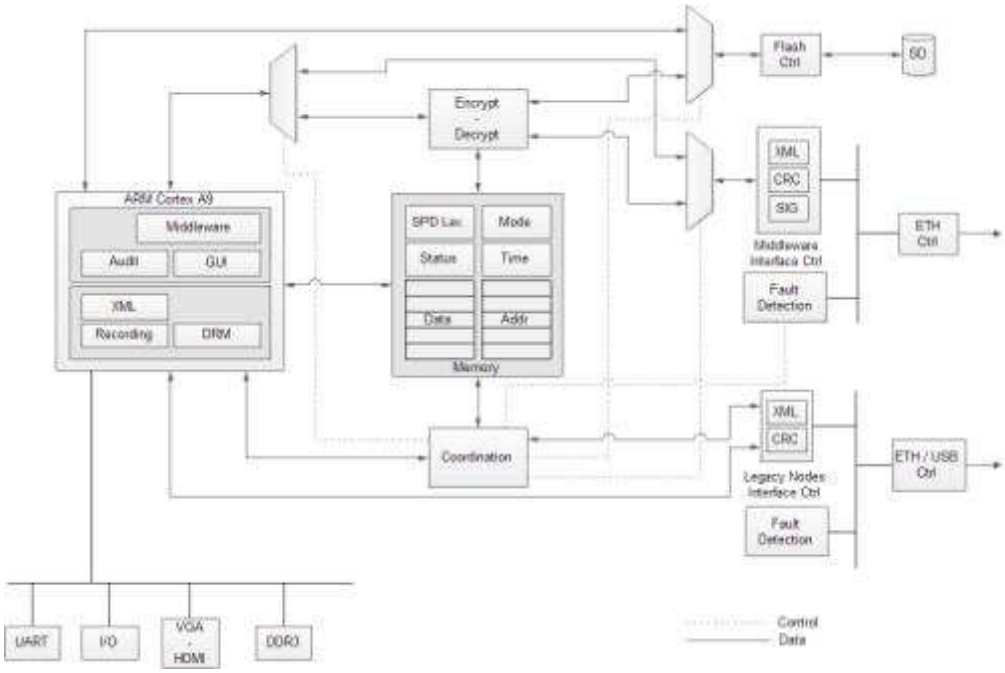


Figure 6-4 nS-ESD-GW HW architecture

As shown in Figure 6-4, the nS-ESD-GW is constituted by several modules hereafter specified:

- **Dual core Cortex ARM A9 in asymmetric multiprocessing (AMP) configuration.** Each processor is configured to run its own software and, in particular, a Linux distribution runs on CPU1 while bare metal applications run, as parallel threads, on CPU2.
- **Encrypt/Decrypt IP core.** This component has been developed as FPGA-based module to assure high flexibility and performances. The presence of this block guarantees the confidentiality and the security of data.
- **Coordination module.** It provides balancing functionality according to the SPD level. To assure the required level of security and dependability, it dynamically adapts its configuration and resources used.
- **Memory.** It stores dynamic data blocks whom contain: the status of OMNIA's components, the status of the nS-ESD-GW, SPD levels received by the nSHIELD Middleware, operational mode and freshness information.
- **Middleware Interface Controller.** It is constituted by different sub-components: interfaces, mechanisms of digital signature check, fault detection and the data integrity. In the context of the avionic scenario, it represents the interface between the nS-ESD-GW and the nSHIELD Middleware through the SDR cognitive radio.
- **Legacy nodes Interface Controller.** Similar to the Middleware Interface controller, this component has been subdivided into sub-modules to manage the data integrity, the fault detection and the messages conversion. In the context of avionic demonstrator, it represents the interface between the nS-ESD-GW and the OMNIA platform.
- **Additional peripherals.** The nS-ESD-GW also offers a set of common ready-to-use interfaces as: UART, general purpose I/O, VGA and HDMI.

As previously mentioned, Cortex ARM A9 processors are in AMP configuration; this mechanism allows to run an operative system and bare metal applications with the possibility of loosely coupling those applications via shared resources. Figure 6-5 depicts the software layered architecture designed.

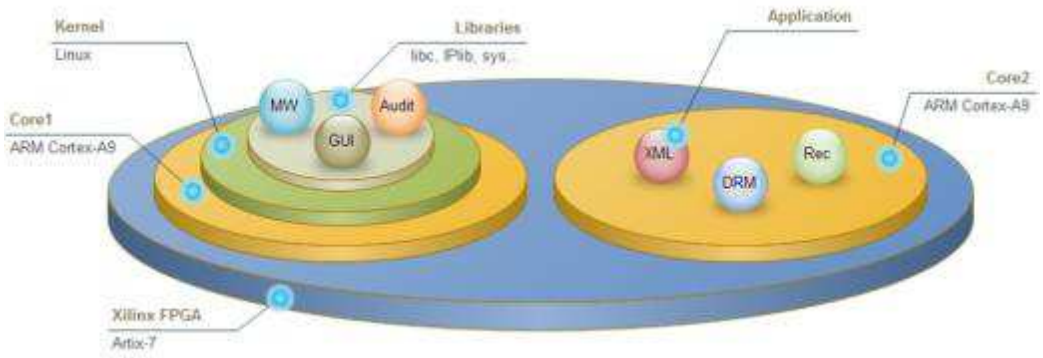


Figure 6-5 nS-ESD-GW SW partitioning

According to this configuration, the Linux operative system is responsible for:

- Audit;
- Graphical user interface for monitoring;
- nSHIELD Middleware management application.

The bare metal applications are responsible for:

- XML parsing and filtering;
- Recording / system dump;
- Dynamic reconfiguration.

6.2.1 nS-ESD-GW Gateway SPD features

The nS-ESD-GW is a component defined into the SHIELD framework. Its scope is to foster the interconnection of legacy nodes building up a SHIELD cluster. As constituted, the cluster will inherit from the ns-ESD-GW several SPD features; in particular the cluster will have:

- **Security:** Mechanism for encrypted communication. The cluster nodes will leverage on encryption/decryption methods provided by nS-ESD-GW to exchange messages with other SHILED components.
- **Security:** Mechanisms for data and message integrity. These mechanisms will ensure the accuracy and the consistency of the exchanged messages.
- **Dependability:** Mechanism for Fault detection. They are obtained by the means of fault tree logic and decision support systems.
- **Dependability:** Mechanism for internal Cluster reconfiguration. Getting information about the nodes status, current faults and context, the system is able to identify a new nodes configuration and eventually apply it to the nodes cluster.

In Figure 6-6 a logical architecture of functionalities provided by the Gateway is outlined.

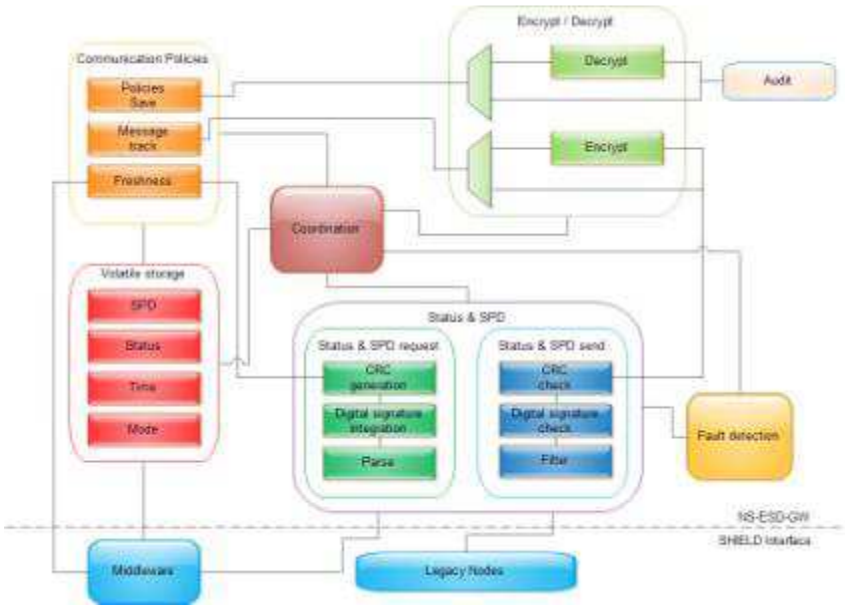


Figure 6-6 nS-ESD-GW Functionalities

The nS-ESD-GW encompasses several communication policies. These policies specify and regulate interrogations (interval, priority, broadcast, unicast, etc), type of messages to be dispatched and/or to be accepted (message alive time, timeout, etc) and the Gateway operational modes. The policies are not upgradeable, avoiding by design the risk of malicious attacks. In accordance to the policies of the nSHIELD framework, the nS-ESD-GW is able to evaluate the SPD level provided by the Middleware and to change its operational mode. This means that the Gateway is able to:

- Increase/decrease the rate of the messages read/write to OMNIA;
- Enable/disable cryptographic modules;
- Increase/decrease the writing of the logfile concerning the audit function.

The nS-ESD-GW encompasses two distinct Data Freshness methods. Stored data can be updated periodically or upon system requests. The update time interval is controlled by an internal configurable register. A dedicated set of registers are used to store information about the data freshness. Specifically, the number representing the amount of time since data have been stored is saved into a specific register of the Gateway and it is available as output.

The confidentiality and the security of private information is ensured through the adoption of encryption/decryption modules for the data writing. A non-volatile memory is used to store long-term data. While the Gateway is running in a secure mode the data processed and algorithms are stored as encrypted. Also, a mechanism of digital signature check is applied during the communication with nodes in order to detect any malicious attempt to open a non-trusted communication channel.

6.3 SPD-driven Smart Transmission Layer

Communication between the UAV and the control center shall be based on utilization of the capabilities of the highly-reconfigurable, computationally unconstrained nSHIELD SDR/Cognitive-capable node, that is, its Smart Transmission Layer functionality, developed within the task T4.1.

Role of the Smart Transmission Layer is providing reliable and efficient communications even in critical channel conditions by using adaptive and flexible algorithms for dynamically configuring and adapting various transmission-related parameters. Namely, for the purposes of the Dependable Avionics demonstrator, the following will be exercised and demonstrated:

- Basic network functionalities:
    - network entry;
    - node authentication;
    - internode communication;
    - topology awareness(number of nodes, mutual visibility, connection, location);
    - reconnection of a node after power cycle/link loss;
    - choice of operating frequencies in accordance with the predefined frequency plan
- Waveform and channel interoperability - by using the appropriate software tools (see Appendix), we shall be able to model different kinds of waveforms and emulate different channel conditions
- Jamming detection and counteraction - By measuring link channel quality, and performing consistency checks between SNR-PER and SNR-location, a decision on whether jamming takes place can be taken. In this case, a security counter-algorithm shall be deployed. The algorithm shall encompass the following functionalities (which of them shall be exercised at a given moment depends on the SPD level imposed by the overlay):
    - moving to a new frequency
    - changing physical or logical waveform parameters, i.e. modulation, bit-rate, transmit power, FEC and MAC protocols
    - selecting a different waveform for transceiving

Basic demonstrability of the secure and dependable communication can be depicted by the following scenario:

1) A wireless node-to-node communication between the two communication modules (one is placed on the vehicle, and one on the ground as the control center) is initiated on a chosen frequency in VHF/UHF band. SPD level is set to maximum (10), meaning that all transmission parameters, such as modulation, channel coding, etc., are set to provide the highest resilience to possible interference. Visualization of the link quality is provided on the control center.
2) Jamming disturbances on the channel that is momentarily used for communication are created using another SDR. Jamming power is gradually increased (with SDRs, this is fairly easy to do on-the-fly), until the control center and the onboard-radio decide that the interference level is too high for the normal communication to continue. Link quality is adequately depicted at the control center (possibility of creating some sort of visual/audio “alert” message).
3) Both radios change their operating frequency according to a pre-defined scheme. Uninterfered communication takes place once more, and the satisfying link quality is restored.
4) SPD level is changed (e.g. to 5). Several transmission parameters may now be changed (e.g. higher-order modulation techniques, reducing no. of transmitted redundant bits, etc.), which typically reduce robustness but allow for a higher data rate.
5) Steps 2) and 3) are repeated.

The hardware platform consists of two cooperating entities. Secure Wideband Multi-role – Single-Channel Handheld Radio (SWAVE HH) is used as a RF front end and as the secondary processing platform, whereas SPD-driven Smart Transmission Layer (OMBRA) v2 is used as the primary processing platform. SWAVE HH and OMBRA v2 can be connected either through a high speed serial connection or through a USB/Ethernet, depending on the required throughput. OMBRA v2 needs to be plugged into a carrier board providing electrical interfaces towards radio and external instrumentation and power supply.
Sketch of the complete platform is as follows:

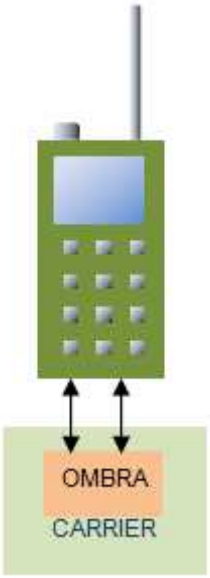


Figure 6-7 SPD-driven Smart transmission layer platform

SWAVE HH is SCA 2.2.2 compatible, and supports reconfiguration of all of its transmission parameters on-the-fly. It is capable of operating in the complete VHF and UHF bands.
OMBRA v2 is a powerful embedded platform equipped with a GPP, DSP and FPGA, being suitable for highly-demanding computational tasks.
More in-depth technical details regarding the SPD-driven Smart Transmission Layer are provided in deliverables D4.2 and D4.3
The interfaces to other relevant components of Avionic System demonstrator are presented in detail in Appendix A.3

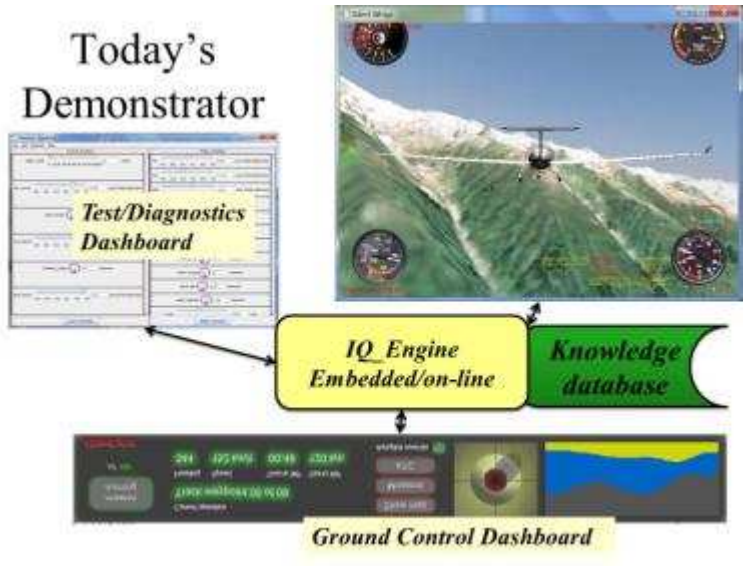
6.4 IQ_Engine Autopilot and Cognitive Pilot

6.4.1 Why IQ_Engine in WP7.3

This chapter aims to describe the working environment for the required components, including hardware and software, for this scenario to work properly according the nSHIELD objective.

The IQ_Engine is a generic knowledge bases system[1], which is based upon a patented technology which is claimed to be in compliance with avionics requirements.

The IQ_Engine can take many roles, due to its flexibility and speed, and in this project it is assigned to a task wich will reside in the ground control station PC (standard Windows environment).



The following paragraph describes its current status, and what will/may be required in the nSHIELD role.

6.4.2 IQ_Engine basic description

The IQ Engine is currently being demonstrated as an Autopilot for a simulated airplane. In the image you see the simulator up in the right hand corner, the IQ_Engine test dashboard to the left, and a simple ground station at the bottom (The rationale being that with a high degree of autonomy, no detailed instruments are needed)

The plane is here set up to fly in a hilly Alpine landscape with heavy winds and turbulence (can be varied). Since the simulator displays many of the instruments (to be selected), there is no need for these in the ground station.

[1] http://en.wikipedia.org/wiki/Knowledge-based_systems

The altitude profile (yellow, blue, black lower right) includes the ground (black) for reference. The view of the simulated aircraft is chosen to see the movements of the controls, but there are various options to see the plane from the cockpit, in 3D, and passing by. By a single click one can see the map with the plane's position, direction and speed.
More about the simulator can be found here[2].

6.4.2.1 Interfaces Inputs/Outputs

From the simulator we receive these inputs (called outputs from the simulator side):

```
unsigned int timestamp;          // Millisec  Timestamp
double position_latitude;        // Degrees   Position latitude,
double position_longitude;       // Degrees           longitude,
float  altitude_msl;             // m         Altitude - relative to Sea-level
float  altitude_ground;          // m         Altitude above gnd
float  altitude_ground_45;       // m         gnd 45 degrees ahead (NOT IMPLEMENTED YET),
float  altitude_ground_forward;  // m         gnd straight ahead (NOT IMPLEMENTED YET).
float  roll;                     // Degrees
float  pitch;                    // Degrees
float  yaw;                      // Degrees
float  d_roll                    // Deg/sec   Roll speed.
float  d_pitch                   // Deg/sec   Pitch speed.
float  d_yaw                     // Deg/sec   Yaw speed.
float  vx                        // m/sec     Speed vector in body-axis
float  vy
float  vz
float  vx_wind                   // m/sec     Speed vector in body-axis, relative to wind
float  vy_wind
float  vz_wind
float  v_eas                     // m/sec     Equivalent (indicated) air speed.
float  ax                        // m/sec2    Acceleration vector in body axis
float  ay
float  az
float  angle_of_attack;          // Degrees   Angle of attack
float  angle_sideslip;           // Degrees   Sideslip angle
float  vario                     // m/sec     TE-compensated variometer.
float  heading                   // Degrees   Compass heading.
float  rate_of_turn              // Deg/sec   Rate of turn.
float  airpressure               // pascal    Local air pressure (at aircraft altitude).
float  density                   // Air density at aircraft altitude.
float  temperature               // Celcius   Air temperature at aircraft altitude.
```

The red-coloured parametres are currently in use. For communication, Silent Wings can output various flight data to a configurable UDP socket. More details are found here[3]. NOTE: "Output" from Silent Wings is "Input" to the IQ_Engine, and vice versa.

6.4.2.2 Commands

To the simulator we send these commands (inputs at the simulator side):

[2] http://www.silentwings.no/

[3] http://wiki.silentwings.no/index.php?title=UDP_Output

AIL <value> - Ailerons, [-1.0 -1.0]
ELE <value> - Elevator, [-1.0 -1.0]
ABK <value> - Airbrakes, [0.0 - 1.0]
FLP <value> - Flaps, [0.0 - 1.0]
RUD <value> - Rudder, [-1.0 - 1.0]
TRM <value> - Elevator trim, [-1.0 - 1.0]
GEAR <value> - Gear [0,1]
WTR <value> - Water valves (int)
THR <value> - Throttle, [0.0 - 1.0]
PIT <value> - Propeller pitch, [0.0 - 1.0]
WBK <value> - Wheel brakes, [0.0 - 1.0]
PBK <value> - Parking brakes [0, 1]
PRBK <value> - Propeller brake [0.0 - 1.0]
IGN <value> - Ignition, [0,1]
MIX <value> - Mixture, [0.0 - 1.0]
MAG <value> - Magnetos, [1,2,3]
FUV <value> - Fuel valve
FUT <value> - Fuel tanks
FUP <value> - Fuel pump, [0.1]
ENST <value> - Engine start [0, 1]
DCMP <value> - Engine decompression [0, 1]
PRM <value> - Fuel primer, [0, 1]
CHT <value> - Carburetor heat, [0.0 - 1.0]
CWFP <value> - Cowling flap [0.0 - 1.0]
EEXT <value> - Engine boom extend, [0, 1]
ERET <value> - Engine boom retract, [0, 1]
TBO <value> - Engine turbo, [0.0 - 1.0]
PANH <value> - Camera pan horizontal (degrees)
PANV <value> - Camera pan vertical (degrees)
ZOOM <value> - Zoom +/-
APHDG <enable> <target> - Auto pilot, heading hold. Enable/disable with 1/0, target in degrees.
APSPD <enable> <target> - Auto pilot, speed hold. Enable/disable with 1/0, target in km/h.
APROL <enable> <target> - Auto pilot, auto bank. Enable/disable with 1/0, target in degrees.
APRUD <enable> - Auto pilot, auto rudder enable/disable, [0, 1] (Coordinated flight).
APSPDC <enable> - Auto pilot, speed command enable/disable, [0, 1].
HUD <mode> - Set HUD: 0 = Off, 1 = Flight mode, 2 = Competition mod.
JOY <enable> - Enable/disable Joystick control, [0,1].
VIEW <mode> - Set view mode: 0 = Cockpit, 1 = Spot, 2 = Fixed, 3 = Flyby, 4 = Free flight, 5 = Chase, 6 = Overview
SRT <val> - Simulation rate: < 0.0 - 64.0]
QUIT - Quit simulation
RST - Restart simulation
PAUSE <enable> - Enable/disable pause
CPT <enable> - Enable/disable cockpit graphics, [0,1]
TRK <enable> - Enable/disable track ribbons, [0.1]
LAB <enable> - Enable/disable labels, [0,1]
PIC - Grab screenshot
TSK <enable> - Enable/disable task graphics, [0,1]
THM <enable> - Enable/disable thermal visuals, [0,1]
VEC <enable> - Enable/disable force vectors, [0,1]
CHAT <message> - Send chat message
MAP - Switch to map screen
FLT - Switch to flight screen

HELP <enable> - Enable/disable help window, [0,1]
PLB <enable> - Enable/disable playback controls (Viewer only)
RSLT <enable> - Enable/disable results window [0,1]
SPLT <enable> - Enable/disable standings plot window [0,1]
AC <number> - Set current aircraft [1, n]
OFS <seconds> - Offset current time by <seconds> (Viewer only).

The red-coloured controls are currently in use. More will be included as more intelligence is added to the IQ_Engine.

As for input, Silent Wings can receive various flight controls to a configurable UDP socket. More details are found here[4]. NOTE: "Output" from Silent Wings is "Input" to the IQ_Engine, and vice versa.

6.4.2.3 Internal data

In addition to the above, the IQ_Engine has its own set of variables, that are constantly updated as new data arrives. These are e.g:

```
  float roll_target; // Degrees
(Which roll position to go to, e.g. 0 degrees or 30 degrees for turning, etc)
  float pitch_target; // Degrees
  float yaw_target; // Degrees

  float roll_difference; // Degrees
What is the difference between the current roll position and the target position?
  float pitch_difference; // Degrees
  float yaw_difference; // Degrees
```

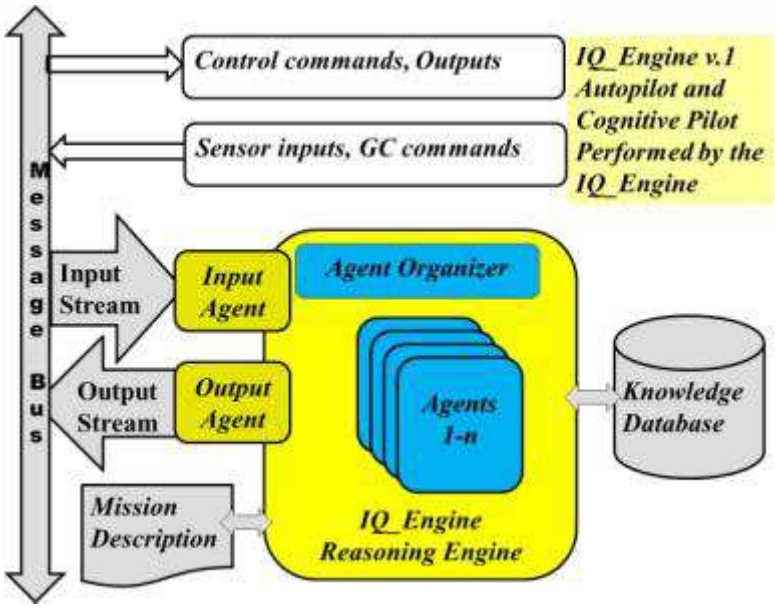
6.4.3 Operating Environment in the WP-7.3 demo

IQ_Engine works as anticipated, and will comply with requirements such as minimum 20 control signals per second (max 50ms between control outputs), often a requirement for unstable aircraft.

The input/output side is easy to change or adapt, and does not affect the content of the knowledge base.

A somewhat more detailed description of the IQ_Engine's internal structure, when it acting like an Autopilot, follows here:

[4] http://wiki.silentwings.no/index.php?title=Remote_Control



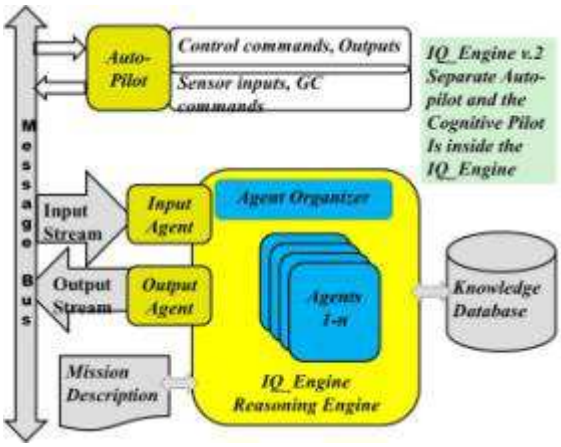
6.4.4 IQ_Engine sensor inputs and command lines

In the nSHIELD set-up, the IQ_Engine will NOT act as an Autopilot, it will rather give the Autopilot Commands on a higher level, such as

- where to go (heading, altitude and velocity),
- what to do when you get there (loiter in circle, take pictures, etc),
- to follow a target on the ground while loitering (position of the target being continuously updated).

All detailed commands, such as setting a throttle speed, setting of rudders, ailerons and elevator wll be managed locally in the UAV by the autopilot). A number of the data listed above in the section (Simulator/Plane data inputs) will be needed, however.

The IQ_Engine is set up to function as an Autopilot as described in the chapter “IQ_Engine current status”. This will be implemented in an environment as similar as possible to the Selex/SES Ground Station environment, and using UDP as means of communications. The implementation is done I Norway, based upon libraries and software sent to Alfatroll from Selex/SES.



The ground station may very well be the one provided by Selex/SES, ***including the Mission define function.***

The purpose us to make sure the operating environment fulfills the requirement for both parties, and that the IQ_Engine can function in conjunction with the intended Ground Control Station.

Proposed Step two: IQ_Engine integration in nSHIELD

The IQ_Engine is set up to function as a Ground Control extension, giving commands to the simulated Autopilot as described in the chapter “IQ_Engine requirements in nSHIELD”.

This will require a message protocol to be determined and put in place, operating in the Selex/SES Ground Station environment, and using nSHIELD protocol layers. The implementation is done I Norway, based upon libraries and software sent to Alfatroll from Selex/SES.

The ground station is provided by Selex/SES, ***including the Mission define function.***

The purpose us to make sure the operating environment fulfills the requirement for both parties, and that the IQ_Engine can function in conjunction with the intended Ground Control Station.

6.4.5 Scenario proposal.

When selecting scenario, it is of vital interest to the nSHIELD project that all the communications links, layers, and components are included for a proper testing of the whole.

One scenario 1 has been suggested:

- Two UAV's are in flight above the work station. One UAV (A) loses its GPS signal, and is unable to determine its position precisely. By using observations from the ground or the other UAV (B) (vision or radar), the exact position is determined, and relayed on a frequent basis to the UAV A with failing GPS.

This scenario may be hard to set up due to the uncertainty with determination of the exact position of the UAV with the failing GPS (Alfatroll's judgement). All components of the signal chain can be involved, though.

One scenario 2 is suggested, which will likewise fulfil the requirements of including all components.

- Two UAV's are airborne, one (A) is responsible for general surveillance, the other (B) is hooked onto a target on the ground. The target may be a person, a car, a house etc, and A is given the task to cover the object whether it is still or moving.
  The GPS positioning af A fails, and B is reassigned to take over for A, until another UAV (C) can be brought in to cover the general surveillance. Everything is controlled by tha ground station by sending appropriate commands to the UAV's.

Both scenarios fulfil the requirements for nSHIELD testing of the complete system, but the scenario 2, since the logic of the autonomous action to be taken by the Alfatroll system in the ground station is more demanding and realistic.

6.4.6 IQ_Engine Autopilot and Cognitive Pilot SPD features

To be supplied.

6.5 Semantic model

The Semantic Model was delivered as a preliminary prototype in D5.2 [1], and is described in detail in D5.3 [2].

The methodology adopted to produce the SHIELD semantic models is based on the definition of two information repository: one for the subtract information and one for the domain dependent information. The first one contains the ontology provided by the components and the second one contains the parameters, relations or rules tailored by the domain experts once the system is deployed.

For the purposes of the demonstrator, two small but significant example of these repositories will be provided.

- For the ontology, an XML template will be distributed to the prototypes providers to be filled with relevant information: this files will be collected by the OSGI and stored in a specific repository (that is not necessarily a Data Base but could be also a portion of RAM).

- For the domain dependent library, a set of entries will be prepared and manually inserted in the OSGI, to be used for control purposes

6.5.1 Semantic model SPD features

The ontology file includes simply a list of SPD functionalities offered by a specific system component. For each SPD functionality, some attributes are reported, in line with the procedures for metrics computation, and these attributes are filled with the metric values elaborated following metrics guidelines (attack surface and/or multimetrics).

The domain dependent library contains mainly: i) a set of inclusion/non-inclusion relations for the different SPD components (if any), ii) the overridden values for the ontology attributes (if any) and iii) policies to force the behaviour of the SHIELD system (if any).

6.6 Multi-metrics

Multi metric approach will address the procedure maintained by document 2.8 called “An Evolutionary-Fuzzy Approach towards Multi-Metric Security Risk Assessment in Heterogeneous System of Systems”. For this specific scenario the scenario owner should reflect the following procedure.

a) Select correct metrics for Railway scenario from document 2.5. This selection takes into account the risks identified in the present document. The following table could be an example of selected metrics:

Threat	Metric
Interferences	nº interferences/total messages
Access control guarantee	cryptographic ley length for AUTH
	nº accesses
Corruption	nº messages altered/total messages
	nº message digest corrupted (MD5/SHA1)

b) An analysis of selected metric should be featured as follows (this is a validation draft that could be extended to more metrics analysis):

Threat	Full domain	Comments	Funtion types in correct region	Function type in incorrect region	Layers	SPD
Interferences	[0, max_{time}]*	unit is relation of nº of messages & inter	linear	logarithmic	Network	S
Access control guarantee	[0,2048]	Key length (129, 192, 256, 512)	linear	logarithmic	Net, midd	P
	[0, max_{time}]	unit is relation of nº of messages	linear	logarithmic	Net, midd	P
Corruption	[0, max_{time}]	unit is relation of nº of messages	linear	logarithmic	Node	D
	[0, max_{time}]	unit is relation of nº of messages	logarithmic	logarithmic	Node	D

c) Expert system development according to scenario owner expertise. In this case, a specific FUZZY-LOGIC IF- THEN algorithm will be issued according to 2.8 document.

d) Final Dashboard should be as follows:

layer	S	P	D
Node	(G,Y,R)	(G,Y,R)	(G,Y,R)
Network	(G,Y,R)	(G,Y,R)	(G,Y,R)
Middleware	(G,Y,R)	(G,Y,R)	(G,Y,R)
Overlay	(G,Y,R)	(G,Y,R)	(G,Y,R)

Summarising, Scenario owners identify key metrics, normalise and analyse them, and finally with the help of a fuzzy IF-THEN algorithm can obtain an aggregated heterogeneous multi metric measurement via this dashboard.

6.7 Surface metric

6.7.1 Surface metrics SPD features

Attach Surface Metric approach starts from the following considerations:

1 a threat is the origin of the fault chain (fault -> errors -> failures) for the dependability concerns and as the potential for abuse of protected assets by the system for security concerns.

2 The attacker is the threat agent, it is a malicious human activity or non malicious event.

3 An attacker uses nSHIELD's entry and exit points to attack the system.

So it was introduced an entry and exit point framework to identify three relevant factors: Porosity, Controls, Limitations.

An entry and exit point contribution to the attack surface reflects factors' likelihood of being used in attacks. For example an entry point running a method with root privilege is more likely to be used in attacks than a method running with non-root privilege. We introduce the notion of a damage potential-effort ratio (der) to estimate porosity contribution.

A system’s attack surface measurement (Actual SPD Level) is the total contribution of the system’s factors along the porosity, controls, and limitation.

Each supplier of a product or system that will be part of this demonstrator must provide the data needed for the calculation of SPD level defined by the adopted metric approach.

These data will be provided by filling in an excel sheet which is being finalized and will contain all the information necessary to Actual SPD level calculation.

The Attack surface metric approach definition and the details of data to be provided are contained in deliverable D2.5 [2] (insert this reference in the reference list [2] nSHIELD, D2.5: Preliminary SPD Metrics Specification)

6.8 Middleware Intrusion Detection System

The Intrusion Detection and Filtering Module was delivered as a preliminary prototype in D5.2 [1], and is described in detail in D5.3 [2].

The IDS prototype operates as a TCP/UDP gateway, having network interfaces connected towards the Middleware services present, and similar network interfaces towards other parts of the system accessing the Middleware services. The main role of the preliminary IDS prototype is to filter requests towards middleware functionality, received from network interfaces that utilize common and/or public network infrastructure, also protecting against requests from compromised nSHIELD nodes. The IDS prototype is created to operate autonomously after initial setup, but it also provides function interfaces for real-time monitoring and control according to SPD requirements, implemented natively in the OSGI Framework environment (see also Chapter 6.10, OSGI middleware).

6.8.1 IDS prototype interfaces

In its current status, the preliminary IDS prototype has generic network interfaces for receiving and forwarding requests that are to be filtered. It is however anticipated that TAP / TUN virtual network interfaces could be used to physically separate and protect internal (Middleware services) and external (other components and networks besides Middleware) network domains. These changes could mostly be implemented in a transparent manner for the system components using middleware services, but may impact how connection methods towards middleware services should be implemented.

The network interfaces operate in a transparent manner, but require setting up network infrastructure so that requests are received by the gateway instead of the middleware services natively. For this purpose, the Intrusion Detection and Filtering Module provides additional function call interfaces towards the middleware services that implement the use of the IDS – see next chapter about SPD features.

6.8.2 IDS prototype SPD features

The preliminary version of the Intrusion Detection and Filtering Module provides the following features for the Middleware services utilizing the IDS. These features are controllable via function interfaces in the Middleware environment in Java:

- Intrusion detection configurable per service
- Provides blacklisting and whitelisting for clients – operation mode and lists can be controlled from the Overlay based on higher level semantic SPD information (e.g. based on trust level associated with clients obtained from Secure Discovery)
- Critical Load Detection of Server
- Can be switched to whitelisted or blacklisted mode, or can switch automatically under critical load (can be controlled according to required SPD level changes as well)
- Provides function interface to query Service Metrics that can be used to assess SPD level of the prototype:
    - *totalIncomingRequestCount*
    - *totalOutgoingResponseCount*
    - *totalDroppedFromQueueCount*
    - *currentQueueSize*
    - *totalBlacklistRejection*
    - *totalWhitelistRejection*

6.9 Protection profile

The nSHIELD project has the ambitious to be a commercial standard for Security, Privacy and Dependability regarding embedded systems. At this purpose the idea of a Protection Profile (at the moment only for middleware layer) is a first step to define a security problem definition and security objectives for embedded systems.

As defined in D5.3, a protection profile (PP) is a Common Criteria (CC) term for defining an implementation-independent set of security requirements and objectives for a category of products, which meet similar consumer needs for IT security. Examples are PP for application-level firewall and intrusion detection system. PP answers the question of "what I want or need" from the point of view of various parties. It could be written by a user group to specify their IT security needs. It could also be used as a guideline to assist them in procuring the right product or systems that suits best in their environment. Vendors who wish to address their customers' requirements formally could also write PP. In this case, the vendors would work closely with their key customers to understand their IT security requirements to be translated into a PP. A government can translate specific security requirements through a PP. This usually is to address the requirements for a class of security products like firewalls and to set a standard for the particular product type.

Protection Profile defines the rules or rather the SPD requirements that must be met by prototypes Integration that make up an embedded system aiming to be SHIELD compliant (as indicated above, at this time are shown only the SPD requirements that the middleware of the system must meet).

6.9.1 Protection Profile SPD features

As indicated in the nSHIELD middleware PP [1] (insert this reference in the reference list … [1] nSHIELD Middleware Protection Profile - nSHIELD Project - PP1.0 - 24.5.2013) Protection Profile (PP) applies to middleware layer of a generic Embedded system which aim to be compliant to nSHIELD project, that we'll consider as Protection Profile Target of Evaluation (TOE).

The TOE is part of a system. It is a software and its purpose is to act as a glue for the different SPD services offered by a nSHIELD compliant embedded system.

The TOE features security functions for:

- Identification & Authentication;
- Auditing;
- Data Integrity;
- Availability.

This generic identification of security functions can be mapped on TOE through the following statement:

- Orchestrator that improves services discovery/composition is able to identify and authenticate services/devices (discovered/composed);
- TOE is able to record security relevant events;
- TOE is able to verify the integrity of composition command definition;
- TOE is able to grant services availability.

6.10 OSGI middleware

The OSGI framework was delivered as a preliminary prototype in D5.2 [1], and is described in detail in D5.3 [2]. This framework is the platform adopted to emulate the functionalities of a generic Middleware. The considerations that lead to this choice are reported in the following and taken directly from the pSHIELD documents, in which this choice was preliminarily investigated. In fact, considering the possible available SOA open solutions, the decision was to select OSGi as the reference service platform to implement the Middleware services. The main reasons leading to this decision are:

- OSGi is an open standard;
- OSGi has a number of open source implementation (Equinox, Oscar, Knopflerfish);
- OSGi can be executed even over lightweight nodes (Embedded Systems Devices);
- OSGi has been implemented using different programming languages (e.g. Java, C, C#);
- The Java implementations of OSGi is fast to deploy and it is much easier to learn, facilitating even an active and collaborative prototype deployment among partners;
- OSGi plugins are available for a number of IDE tools (i.e. Eclipse, Visual Studio, etc.);
- OSGi can be easily deployed in Windows (XP, 7, Mobile), Linux, MAC and Google (Android) OSes.

More in particular it has been decides to use the open source Knopflerfish OSGi service platform. Knopflerfish (hereafter referred as to KF) is a component-based framework for Java in which units of resources called bundles can be installed. Bundles can export services or run processes, and have their dependencies managed, such that a bundle can be expected to have its requirements managed by the container. Each bundle can also have its own internal classpath, so that it can serve as an independent unit, should that be desirable. All of this is standardized such that any valid Knopflerfish bundle can be installed in any valid OSGi container (Oscar, Equinox or any other).
Basically, running OSGi is very simple: one grabs one of the OSGi container implementations (Equinox, Felix, Knopflerfish, ProSyst, Oscar, etc.) and executes the container's boot process, much like one runs a Java EE server. Like Java EE, each container has a different startup environment and slightly different capabilities. The KF environment can be downloaded here: http://www.knopflerfish.org/

The KF start-up environment is shown below:

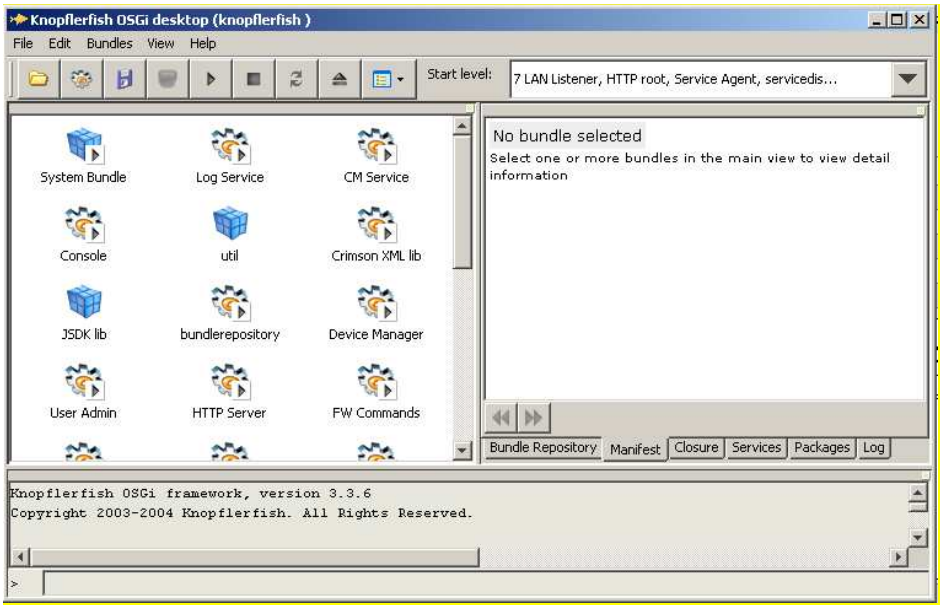


Figure 6-8 Knopflerfish start-up environment

6.10.1 OSGI middleware SPD features

One of the most important peculiarities of the KF OSGi is that it already offers a standard orchestration environment that, once correctly setup, can act as the SHIELD Orchestration Core SPD Service. Thus the Orchestration functionalities comes for free when using an OSGi framework, instead of using other SOA implementations.

The prototype architecture derives directly from the architecture described in the previous section. Each SHIELD Middleware component is mapped into an OSGi bundle and, when needed, decoupled into a composition of interoperating bundles each providing a specific functionality. This modular approach simplify the design, development and debugging of the whole system. Even the Innovative SPD Functionalities have been implemented as OSGi bundles. Each OSGi bundle has its own dependencies, provides a set of functionalities, requires a set of functionalities and is characterized by a specific SPD level. Each bundle can be registered in the Service Registry to advertise itself, to maintain updated its status in order to be discovered. Each bundle can also store its description in the Semantic Database, to be semantically composed. Each bundle interfaces the rest of the architecture providing a set of functionalities and requiring a set of functionalities, exactly as a software component does. More in particular each bundle is decoupled into two parts: the interfacing part (API) and its implementation part (IMPL). This separation between API and IMPL ease the substitution at runtime of a specific bundle, to change from one implementation to another. This substitution can be due, as an example, to the necessity to strengthen the SPD level of a specific functionality.

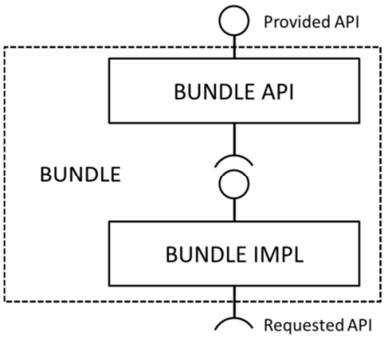


Figure 6-9 Bundle architecture

This framework will be installed on a Laptop and interconnected to the Gateway to drive the security functions. At a preliminary analysis, the interaction will be done by means of Ethernet interfaces and through the Intrusion Detection Bundle, that is just in the middle between the final system and the OSGI.

6.11 Control Algorithms

The Control Algorithms was delivered as a preliminary prototype in D5.2 [1], and is described in detail in D5.3 [2]. However the version of the control algorithms that will be included in the demonstrator will most likely be the one delivered with the final prototypes, since it will be fully compliant with the mechanism for metrics computations. The control algorithms work as follows: at first a set of candidate technologies is identified as well as the SPD value desired by the user; then an algorithm is applied (a simple optimization, an extensive graph search, a model driven control, ecc) whose result is a list of components that should be activated to reach the desired objectives. Finally these solutions are filtered by including the domain constraints/tailoring and, if there are no changes in the metric value, the solution is confirmed, however it is reiterated taking into account the new inputs.

6.11.1 Control algorithms SPD features

The control algorithms are implemented by the Security Agent in the Overlay layer. On a practical point of view, two solutions are envisaged to include them in the demonstrator, depending on the maturity level reached by the implementation task: direct inclusion in the OSGI source code, or interfacing with an external computation platform (i.e. Matlab) that provides the problem solutions. In both these cases the interfaces between the control algorithms and the system are internal and can be easily managed

7 nSHIELD Dependable Avionic use cases

The application scenario for the nSHIELD project is the Avionic System for a Unmanned Aircraft System (the demonstration will use prototype or laboratory equipment) used for a surveillance application.

To preserve the main functionalities represents an example application of great interest for theAvionic System. In particular, in this use case, the following requirements have to be fulfilled:

- Secure and dependable handling of on-board computing and sensor processing capabilities
- Secure and dependable Ground Operator sensor control terminals

The nSHIELD philosophy will be applied to preserve the data for the following components

- AIR/GROUND DATA TERMINAL: the nSHIELD solution will guarantee that the data exchanged between the UAV and Ground Station will be preserved by anomalous interface.
- Avionics Unit : the nSHIELD solution will guarantee that the data acquired by sensors are protected against the possible corruption.
- Mission System: the nSHIELD solution will define solution for easy integration of new sensors and/or replaced old version
- Ground Control System: the nSHIELD solution will be able to manage the different access to the Unmanned System for the Mission Operators and Pilot

8 Dependable Avionic System demonstrator integration

According to the architectural description above, the integration of the nS-ESD-GW gateway in the OMNIA platform is needed with the aim to guarantee the link between the Avionic and the SHIELD worlds. It will be possible to merge these two technologies developing the gateway in accordance to the IMA standard.

Also, being the IQ_Engine the mission supervisor, it must be integrated into the IMA architecture in order to ensure the proper level of SPD in respect of SHIELD and Avionic Demonstrator needs.

To ensure the communication between nodes and middleware according the nSHIELD addressed standards, it will be used the SPD-driven Smart Transmission Layer board. The integration of this node into the whole system is foreseen.

The integration methods and environment, the tools and platforms, integration tests and schedule foreseen will be described in the next issue of the present document.

9 Dependable Avionic System demonstrator validation and verification

9.1 Validation and Verification methods

The IEEE Guide to the Project Management Body of Knowledge[5] defines validation and verification as follows:

- "Validation. The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification."
- "Verification. The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation."

The proposed method of Validation and verification is applying relevant parts of the V-Model of project management used by several national standards [5]. This model clearly identifies the distinct steps during all stages of the development process – from requirements and design specification to implementation, testing, and operation.

In the case of nSHIELD project, the following methodology is proposed. Activities are identified together with deliverables relevant to that stage, where outcomes are described. As per the V-Model, the overall architecture must be designed to be testable. Specifically, all design elements and acceptance tests must be traceable to design requirements – similarly, each design requirement must be addressed by at least one design element and a corresponding acceptance test.

[5] IEEE Guide--Adoption of the Project Management Institute (PMI®) Standard A Guide to the Project Management Body of Knowledge (PMBOK® Guide)--Fourth Edition". p.452. doi:10.1109/IEEESTD.2011.6086685. (http://ieeexplore.ieee.org/servlet/opac?punumber=6086683)

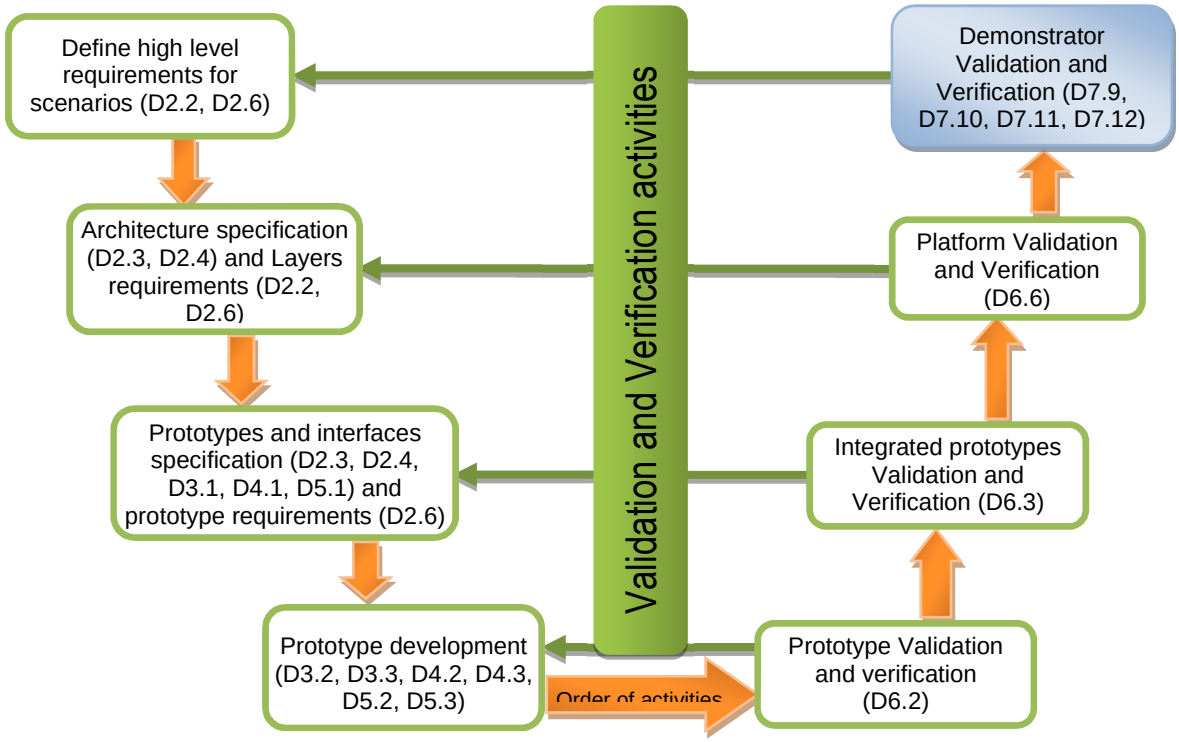


Figure 9-1: Validation and verification activities

Demonstrator Validation and Verification plan (the scope of the current document) deals with the following activities:

- Validation of presented demonstrator scenarios against High level requirements for scenarios – ensuring that the original requirements were adequately covered and demonstrated by use cases.
  Stakeholders involved in Validation and Verification activities:
  - Demonstrator owner as responsible for requirements and scenarios

  Means of validation / verification:
  - Analysis of requirements vs. Demonstrator scenarios

  Justification of prototype, integrated prototype, and platform level validation and verification results in the scope of the current Demonstrator – ensuring that the Demonstrator components were developed as planned and all the lower level requirements were properly implemented.
  Means of validation / verification:
  - Analysis of former (lower-level) validation and verification results, tracing results from prototypes and relevant functionality
- Verification of Demonstrator scenario execution – verifying that the specified use cases (derived from Demonstrator Owner use cases relevant to high level requirements) were executed according to specifications.
  Stakeholders involved in Validation and Verification activities:
  - Demonstrator owner and partner(s) responsible for scenarios
  - Prototype owners and integrators responsible for implementation of relevant functionality

  Means of validation / verification:
  - Comparison of Test results vs. description of scenario (expected behaviour)

The following chapters list information that is available at the current design and development stage about the activities as listed above. The Validation and Verification results will be described in detail in the nSHIELD deliverable D7.11 Dependable Avionic System demonstrator - Validation and Verification Report.

9.2 Validation of demonstrator scenarios

Insert table to list scenario requirements from D2.2 chapter 5.1, and a requirements traceability matrix to show relevance of Scenarios under chapter 7.X w.r.t. high-level requirements (which scenario steps will validate which requirements).

If certain High Level Requirements for Scenarios from [4] are not demonstrated by the Dependable Avionic System Demonstrator Scenarios, please describe rationale (e.g. analogous functionality was demonstrated by other Demonstrators).

9.2.1 Scenario n.1

(Description of relevant requirements from [4])

9.2.2 Scenario n.2

(Description of relevant requirements from [4]), e.g.

High Level Requirements for Scenario	Use case steps (from Chapter 7)	Description
REQ_AVXX	Scenario X steps Y	

9.3 Justification based on prototype and platform Validation and Verification

9.3.1 Validation and verification results for prototypes

Summarize how results of D6.2 validate that prototypes included in current demonstrator are fit for the purpose of the demonstrator.

The description of each component should show which of the original D2.6 requirements have been fulfilled, possibly providing justification for the requirements that were not.

9.3.1.1 OMNIA

(reference to relevant validation and verification results for the prototype)

9.3.1.2 Gateway

(reference to relevant validation and verification results for the prototype)

9.3.1.3 SPD-driven Smart Transmission Layer

See chapter 4.3.4 in nSHIELD, D6.2: Prototype validation and verification Plan [3].

Relevant requirements being validated from nSHIELD, D2.2: Preliminary System Requirements and Specifications [4]:

- REQ_NW01 Confidentiality
- REQ_NW02 Integrity
- REQ_NW04 Fault Tolerance
- REQ_NW05 Self-Management and Self-Coordination
- REQ_NW07 Availability
- REQ_NW13 Fault Recovery
- REQ_NW16 Reliable Transmission Methodologies
- REQ_NW19 Application-Based Configurability

9.3.1.4 IQ_Engine Autopilot and Cognitive Pilot

(reference to relevant validation and verification results for the prototype)

9.3.1.5 Semantic model

See chapter 6.3.1 in nSHIELD, D6.2: Prototype validation and verification Plan [3].

Relevant requirements being validated from nSHIELD, D2.2: Preliminary System Requirements and Specifications [4]:

- REQ_MW6 Information retrieving
- REQ_MW9 Data management
- REQ_SH16 Data backup
- REQ_SH17 Data storage redundancy
- REQ_SH18 Data storage integrity
- REQ_SH19 Data storage confidentiality

9.3.1.6 Multi-metrics

(reference to relevant validation and verification results for the prototype)

9.3.1.7 Surface metric

(reference to relevant validation and verification results for the prototype)

9.3.1.8 Middleware Intrusion Detection module

See chapter 5.3.5 in nSHIELD, D6.2: Prototype validation and verification Plan [3].

Relevant requirements being validated from nSHIELD, D2.2: Preliminary System Requirements and Specifications [4]:

- REQ_SH02 Information transmission integrity
- REQ_SH33 Automated testing tools
- REQ_MW7 Information filtering for intrusion detection
- REQ_MW17 Configurations selection

The relevant high-level requirements towards IDS prototype for the Dependable Avionics System Demonstrator from the above list are REQ_SH02 and REQ_SH33.

9.3.1.9 Protection profile

(reference to relevant validation and verification results for the prototype)

9.3.1.10 OSGI middleware

No specific requirements have been foreseen for the OSGI framework, since it is a consolidate heritage of the pSHIELD project, so there was no need to specify it again as a precondition

9.3.1.11 Control Algorithms

See chapter 6.3.4 in nSHIELD, D6.2: Prototype validation and verification Plan [3]

Relevant requirements being validated from nSHIELD, D2.2: Preliminary System Requirements and Specifications [4]:

- REQ_MW15 Configurations definition
- REQ_MW16 Configurations quantification
- REQ_MW17 Configurations selection

9.3.2 Validation and verification results for integrated prototypes

Summarize how results of D6.3 validate that prototypes included in current demonstrator will be fit for the purpose of the demonstrator.

9.3.3 Platform validation and verification results

Summarize how results of D6.6 validate that prototypes included in current demonstrator will be fit for the purpose of the demonstrator.

Insert table to list all prototypes from chapter 6.X and their relevant linkage to Scenarios in chapter 7.X [Which scenario steps will validate high level functions of which prototypes]

9.4 Verification of Demonstrator scenario execution

9.4.1 Tools and platforms for execution of Demonstrator scenarios

Please list tools and platforms used in verification of the Demonstrator scenarios.

9.4.1.1 Validation and Verification tools for IDS prototype

Chapter 5.3.5 in nSHIELD, D6.2: Prototype validation and verification Plan [3] lists validation steps that are carried out using code for automated testing of Intrusion Detection and Filtering Module:

- Basic functionality test (SendReceiveTest)
- load generation (CriticalLoadTest)
- Information filtering tests (BlackListTest and WhiteListTest)

The testing code referred here is described in [2] and included as source code in [1]. These test cases are available for use in the Railway Demonstrator scenarios developed, or may be used as templates to implement Validation and Verification test cases executed automatically in the Scenarios.

9.4.2 Other HW and SW resources for execution of Demonstrator scenarios

Please list resource to be used in verification of the Demonstrator scenarios, such as measurement, automated testing and input generation, logging, etc. tools, devices, and software.

- IDS prototype uses logging functionality provided by the OSGI Framework (Java package ‘org.knopflerfish.log’)

10 Conclusions

In this deliverable, we presented the integration and validation plan for the Dependable Avionic System scenario demonstrator.

Starting from the description of the dependable avionic scenario, we provided an overview of the involved technologies and we illustrated the solution proposed by the prototypes with their SPD functionalities.

Finally, the document presents the integration, verification and validation approach, in order to demonstrate and validate the nSHIELD framework in the reference application.

The next issue of document will be improved with the detailed description of interface exposed by each component and the clarification of the integration activities.

11 References

[1] nSHIELD, D5.2: Preliminary SPD Middleware and Overlay technologies prototype

[2] nSHIELD, D5.3: Preliminary SPD Middleware and Overlay technologies prototype Report

[3] nSHIELD, D6.2: Prototype validation and verification Plan

[4] nSHIELD, D2.2: Preliminary System Requirements and Specifications

[5] V-Model - http://en.wikipedia.org/wiki/V-model

Appendix A Interface Control Documents

A.1 Interface Control Document OMNIA

The OMNIA prototype is capable to exchange data, acquired from the aircraft sensor, in terms of descrete and digital signals, ARINC 429 data, RS422 and 1553 messages.

The interfaced exposed by OMNIA will be provided in the next set of documents.

A.1.1 Introduction

A.1.2 Protocol Formats

A.1.3 OMNIA – nSHIELD Data Interchange

A.1.3.1 OMNIA Message Format

Message Types

Destination/Source IDs

A.1.3.2 Message Type Description

A.2 Interface Control Document Gateway

The interfaced exposed by the Gateway will be provided in the next set of documents.

A.2.1 Introduction

A.2.2 Protocol Formats

A.2.3 nS-DI – nSHIELD Data Interchange

A.2.3.1 nS-DI Message Format

Message Types

Destination/Source IDs

A.2.3.2 Message Type Description

A.3 SPD-driven Smart Transmission Layer

A.3.1 Introduction

This Interface Control Document (ICD) specifies the interfaces between the Smart Transmission Layer and other relevant modules, as to be in compliance with requirements specified for the Dependable Avionic Scenario.

A.3.2 Protocol Formats

Peer 1	Peer 2	Protocol	PHY	Rate
Smart Transmission Layer	nS-ESD-GW	nS-DI	Ethernet	10/100 Mbps

A.3.3 nS-DI – nSHIELD Data Interchange

The nSHIELD Data Interchange (nS-DI) is used for communications between the SPD-driven Smart Transmission Layer board (OMBRA) and the nSHIELD Gateway (nS-ESD-GW). nS-DI is a full duplex protocol designed to handle asynchronous reads and writes from both parties. nS-DI does not specify any means of control flow, ACKs or NACKs, or command/response. These features must be implemented at the application layer.

The nS-DI protocol is based on Simple Object Access Protocol (SOAP). SOAP is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on XML Information Set for its message format, and usually relies on other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission. Within this scenario, nS-DI protocol uses SOAP-over-UDP standard covering the publication of SOAP messages over UDP transport protocol, providing for One-Way and Request-Response message patterns.

A.3.3.1 nS-DI Message Format

A nS-DI SOAP message is an ordinary XML document containing the following elements:

- A required Envelope element that identifies the XML document as a SOAP message.
- An optional Header element that contains header information
- A required Body element that contains call and response information.

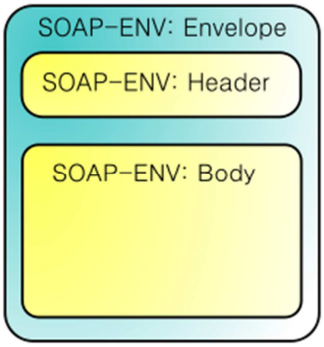


Figure 11-1 Generic SOAP message structure

Message Types

Possible messages are summarized in the following table. These messages are received and interpreted by the Smart Transmission Layer hardware platform.

Message Type	Name	Sink
Control	turnOn	Smart Transmission Layer (OMBRA)
Control	turnOff	Smart Transmission Layer(OMBRA)
Control	setRadioID	Smart Transmission Layer (OMBRA)
Control	setTxPower	Smart Transmission Layer (OMBRA)
Control	setCarrierFrequency	Smart Transmission Layer (OMBRA)
Control	setWaveform	Smart Transmission Layer (OMBRA)
Control	setCrypto	Smart Transmission Layer (OMBRA)
Control	setCryptoKey	Smart Transmission Layer (OMBRA)
TX	sendData	
RX		
Status		

A.3.3.2 Message Type Description

A.3.3.2.1 Common nS-DI response

The majority of the functions from the SPD-driven Smart Transmission Layer server return a common response format.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ombraRemote:errorResponse>
   <m-eErrorCode>OR-SUCCESS</m-eErrorCode>
   <m-sErrorString></m-sErrorString>
  </ombraRemote:errorResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The errorCode field is an enumeration and the value can be OR-SUCCESS (if the function succeded), OR-FAILED (if the function failed) or OR-TOBEIMPLEMENTED (if the function is not implemented yet.

The sErrorString field can contain a human readable string specifying an internal message from the server.

A.3.3.2.2 status()

This section shows the XML format for the request corresponding to the status function and the related response from the Smart Transmission Layer (OMBRA) server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ombraRemote:status>
  </ombraRemote:status>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This function has no parameters. The response is in the common nS-DI format.

A.3.3.2.3 turnOn()

This section shows the XML format for the request corresponding to the turnOn function and the related response from the Smart Transmission Layer (OMBRA) server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
  <ombraRemote:turnOn>
  </ombraRemote:turnOn>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This function has no parameters. The response is in the common nS-DI format.

A.3.3.2.4 turnOff()

This section shows the XML format for the request corresponding to the turnOff function and the related response from the Smart Transmission Layer (OMBRA) server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ombraRemote:turnOff>
  </ombraRemote:turnOff>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This function has no parameters. The response is in the common nS-DI format.

A.3.3.2.5 setRadioID()

This section shows the XML format for the request corresponding to the setRadioID function and the related response from the Smart Transmission Layer (OMBRA) server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ombraRemote:setRadioID>
   <radioID></radioID>
  </ombraRemote:setRadioID>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This function has the following parameter:

- radioID: is an alphanumeric string containing the identifier for the radio.

The response is in the common nS-DI format.

A.3.3.2.6 setTxPower()

This section shows the XML format for the request corresponding to the setTxPower function and the related response from the Smart Transmission Layer (OMBRA) server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ombraRemote:setTxPower>
   <power>0</power>
  </ombraRemote:setTxPower>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This function has the following parameter:

- power: is an integer containing the transmission power level to be set on the radio.

The response is in the common nS-DI format.

A.3.3.2.7 setCarrierFrequency()

This section shows the XML format for the request corresponding to the setCarrierFrequency function and the related response from the Smart Transmission Layer (OMBRA) server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ombraRemote:setCarrierFrequency>
   <tx>0</tx>
   <rx>0</rx>
  </ombraRemote:setCarrierFrequency>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This function has the following parameters:

- tx: is an integer containing the transmission carrier frequency to be set on the radio;
- rx: is an integer containing the receiving carrier frequency to be set on the radio.

The response is in the common nS-DI format.

A.3.3.2.8 setWaveformType ()

This section shows the XML format for the request corresponding to the setWaveformType function and the related response from the Smart Transmission Layer (OMBRA) server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ombraRemote:setWaveformType>
   <param-1>WF-1</param-1>
  </ombraRemote:setWaveformType>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This function has the following parameter:

- param-1: is an enumeration containing waveform id to be set on the radio. Currently, possible values are WF_1, WF_2 or WF_3.

The response is in the common nS-DI format.

A.3.3.2.9 setCrypto ()

This section shows the XML format for the request corresponding to the setCrypto function and the related response from the Smart Transmission Layer (OMBRA) server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ombraRemote:setCrypto>
   <param-2>CRYPTO-OFF</param-2>
  </ombraRemote:setCrypto>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This function has the following parameter:

- param-2: is an enumeration containing the type of cryptography to be enabled on the radio. Currently, possible values are CRYPTO_OFF, CRYPTO_HH_1 (for default device cryptography) or CRYPTO_ELL_CURVE (for elliptic curve cryptography).

The response is in the common nS-DI format.

A.3.3.2.10 setCryptoKey ()

This section shows the XML format for the request corresponding to the setCryptoKey function and the related response from the Smart Transmission Layer (OMBRA) server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote">
 <SOAP-ENV:Body  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <ombraRemote:setCryptoKey>
   <key></key>
  </ombraRemote:setCryptoKey>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This function has the following parameter:

- key: is an alphanumeric string containing the cryptography key to set on the radio.

The response is in the common nS-DI format.

A.3.3.3 WSDL for Ombra service

The following is the WSDL (Web Services Description Language) file describing the functionality offered by the Smart Transmission Layer (OMBRA) web service:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ombra"
 targetNamespace="http://localhost/ombra.wsdl"
 xmlns:tns="http://localhost/ombra.wsdl"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ombraRemote="urn:ombraRemote"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:HTTP="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>

 <schema targetNamespace="urn:ombraRemote"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ombraRemote="urn:ombraRemote"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <simpleType name="eErrorCode"><!-- ombraRemote__eErrorCode -->

   <restriction base="xsd:string">
    <enumeration value="OR-SUCCESS"/><!-- ombraRemote__eErrorCode::OR_SUCCESS -->
    <!-- = 0 -->
    <enumeration value="OR-FAILED"/><!-- ombraRemote__eErrorCode::OR_FAILED -->
    <!-- = 1 -->
    <enumeration value="OR-TOBEIMPLEMENTED"/><!-- ombraRemote__eErrorCode::OR_TOBEIMPLEMENTED
-->
    <!-- = 2 -->
   </restriction>
  </simpleType>
  <simpleType name="eCryptoType"><!-- ombraRemote__eCryptoType -->

   <restriction base="xsd:string">
    <enumeration value="CRYPTO-OFF"/><!-- ombraRemote__eCryptoType::CRYPTO_OFF -->
    <!-- = 0 -->
    <enumeration value="CRYPTO-HH-1"/><!-- ombraRemote__eCryptoType::CRYPTO_HH_1 -->
    <!-- = 1 -->
    <enumeration value="CRYPTO-ELL-CURVE"/><!-- ombraRemote__eCryptoType::CRYPTO_ELL_CURVE -->
    <!-- = 2 -->
   </restriction>
  </simpleType>
  <simpleType name="eWaveformType"><!-- ombraRemote__eWaveformType -->

   <restriction base="xsd:string">
    <enumeration value="WF-1"/><!-- ombraRemote__eWaveformType::WF_1 -->
    <!-- = 0 -->
    <enumeration value="WF-2"/><!-- ombraRemote__eWaveformType::WF_2 -->
    <!-- = 1 -->
    <enumeration value="WF-3"/><!-- ombraRemote__eWaveformType::WF_3 -->
    <!-- = 2 -->
   </restriction>
  </simpleType>
 </schema>

</types>
```

```
<message name="status">
</message>

<message name="errorResponse">
 <part name="m-eErrorCode" type="ombraRemote:eErrorCode"/><!--
ombraRemote__status::m_eErrorCode -->
 <part name="m-sErrorString" type="xsd:string"/><!-- ombraRemote__status::m_sErrorString -->
</message>

<message name="turnOn">
</message>

<message name="turnOff">
</message>

<message name="setRadioID">
 <part name="radioID" type="xsd:string"/><!-- ombraRemote__setRadioID::radioID -->
</message>

<message name="setTxPower">
 <part name="power" type="xsd:int"/><!-- ombraRemote__setTxPower::power -->
</message>

<message name="setCarrierFrequency">
 <part name="tx" type="xsd:int"/><!-- ombraRemote__setCarrierFrequency::tx -->
 <part name="rx" type="xsd:int"/><!-- ombraRemote__setCarrierFrequency::rx -->
</message>

<message name="setWaveformType">
 <part name="param-1" type="ombraRemote:eWaveformType"/><!--
ombraRemote__setWaveformType::_param_1 -->
</message>

<message name="setCrypto">
 <part name="param-2" type="ombraRemote:eCryptoType"/><!-- ombraRemote__setCrypto::_param_2 --
>
</message>

<message name="setCryptoKey">
 <part name="key" type="xsd:string"/><!-- ombraRemote__setCryptoKey::key -->
</message>

<portType name="ombraPortType">
 <operation name="status">
  <documentation>Service definition of function ombraRemote__status</documentation>
  <input message="tns:status"/>
  <output message="tns:errorResponse"/>
 </operation>
 <operation name="turnOn">
  <documentation>Service definition of function ombraRemote__turnOn</documentation>
  <input message="tns:turnOn"/>
  <output message="tns:errorResponse"/>
 </operation>
 <operation name="turnOff">
  <documentation>Service definition of function ombraRemote__turnOff</documentation>
  <input message="tns:turnOff"/>
  <output message="tns:errorResponse"/>
 </operation>
 <operation name="setRadioID">
  <documentation>Service definition of function ombraRemote__setRadioID</documentation>
  <input message="tns:setRadioID"/>
  <output message="tns:errorResponse"/>
 </operation>
 <operation name="setTxPower">
  <documentation>Service definition of function ombraRemote__setTxPower</documentation>
  <input message="tns:setTxPower"/>
  <output message="tns:errorResponse"/>
 </operation>
 <operation name="setCarrierFrequency">
  <documentation>Service definition of function
ombraRemote__setCarrierFrequency</documentation>
  <input message="tns:setCarrierFrequency"/>
  <output message="tns:errorResponse"/>
```

```
 </operation>
 <operation name="setWaveformType">
  <documentation>Service definition of function ombraRemote__setWaveformType</documentation>
  <input message="tns:setWaveformType"/>
  <output message="tns:errorResponse"/>
 </operation>
 <operation name="setCrypto">
  <documentation>Service definition of function ombraRemote__setCrypto</documentation>
  <input message="tns:setCrypto"/>
  <output message="tns:errorResponse"/>
 </operation>
 <operation name="setCryptoKey">
  <documentation>Service definition of function ombraRemote__setCryptoKey</documentation>
  <input message="tns:setCryptoKey"/>
  <output message="tns:errorResponse"/>
 </operation>
</portType>

<binding name="ombra" type="tns:ombraPortType">
 <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="status">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="turnOn">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="turnOff">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="setRadioID">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="setTxPower">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
```

```
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="setCarrierFrequency">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="setWaveformType">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="setCrypto">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="setCryptoKey">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
     <SOAP:body use="encoded" namespace="urn:ombraRemote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
</binding>

<service name="ombra">
 <documentation>Remote Communication Interface for ombra</documentation>
 <port name="ombra" binding="tns:ombra">
  <SOAP:address location="http://localhost:31000"/>
 </port>
</service>

</definitions>
```