Project no: 269317

nSHIELD

new embedded Systems arcHItecturE for multi-Layer Dependable solutions

Instrument type: Collaborative Project, JTI-CP-ARTEMIS

Priority name: Embedded Systems

D2.3: Preliminary System Architecture Design

Due date of deliverable: M9 –2012.05.30

Actual submission date: M12 -2012.08.03

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

Organisation name of lead contractor for this deliverable:

Hellenic Aerospace Industry, HAI

Revision [Issue 14]

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**		
Nikolaos Priggouris	HAI		
Nikos Pappas	HAI		
Hans Thorsen	T2D		
Christian Gehrmann	SICS		
Lorena de Celis	AT		
Jacobo Domínguez	AT		
Andrea Fiaschetti	UNIROMA1		
Renato Baldelli	SE		
Harry Manifavas	TUC		
Konstantinos Rantos	TUC		
Alexandros Papanikolaou	TUC		
Konstantinos Fysarakis	TUC		
Georgios Hatzivasilis	TUC		
Vincenzo Suraci	UNIROMA1		
Reviewed by			
Name	**Company**		
Approved by			
Name	**Company**		

Modification History

Issue	Date	Description
Draft A	22.03.2012	First issue for comments, Table of Contents
Issue 1	14.05.2012	Comments from ACO, TUC, ATHENA and SICS
Issue 2	06.06.2012	Terms, Methodology and Architecture from HAI
Issue 3	22.06.2012	Modifications from HAI in ToC after phone call discussions
Issue 4	06.07.2012	1st version of Network Layer Architecture by HAI
Issue 5	06.07.2012	Suggestion for slightly modified architecture definitions and approach, SICS and T2D
Issue 6	11.07.2012	1st version of Node Layer by ACORDE
Issue 7	13.07.2012	2nd of Node Layer Functionalities by ACORDE
Issue 8	17.07.2012	Definition of Overlay by SE, UNIROMA1
Issue 9	20.07.2012	Update Node Layer section by ACORDE, Update architecture definition strategy by proposing a 4 view approach for each layer Preliminary information regarding interfaces sections
Issue 10	23.07.2012	Incorporate part of TUC contribution
Issue 11	26.07.2012	Section 5 additions/modifications
Issue 12	27.07.2012	Update of section 7 (interfaces)
Issue 13	30.07.2012	Incorporate comments /corrections from ACORDE
Issue 14	03.08.2012	Middleware & Overlay contributions added, Final Version

Contents

Figures

Tables

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

D2.3 is the first of the three deliverables concerning System Architecture (the other two are D2.4 and D2.7) inside nSHIELD WP2, “Scenarios, requirements and system design”. It is an internal deliverable and as denoted by its title, the main objective is to set the framework for the description of nSHIELD System Architecture. According to nSHIELD DoW’s specifications, D2.3 prepares the way for the definition of a formal and conceptual overall system architecture, to address Security, Privacy and Dependability (SPD) in the context of Embedded Systems (ESs) as “built in” rather than as “add-on” functionalities, proposing and perceiving with this strategy the first step towards SPD certification for future ESs. The methodology for achieving this is prescribed in chapter 4, but in a few words the procedure incorporates taking into account and further process aspects such as SHIELD terminology framework, basic requirements, pSHIELD background, metrics, technology status and use cases.

2 Introduction

The main goal of nSHIELD is to advance evolution of technology in the production of Embedded Systems (ESs), ensuring that security, privacy and dependability (SPD) can be strengthened in the context of integrated and interoperating heterogeneous services, applications, systems and devices. To this end, the definition of functional system architecture is one of the most critical project tasks.

nSHIELD Architecture will be comprised by those modules that will implement SPD functionalities and provide SPD services. It is desirable that these services are transparently (to the external user) embedded in the four described layers, the coordination of which is presupposed for the successful discovery, orchestration and provision of functionalities. In particular, the correspondence between the aforementioned layers and the functionalities mostly implemented by SPD modules are:

- At node layer, intelligent hardware and firmware SPD
- At network layer, secure, trusted, dependable and efficient data transfer based on self-configuration,
- self-management, self-supervision and self-recovery
- At middleware layer, secure and efficient resource management, inter-operation among heterogeneous networks
- At overlay layer, composability

The document's structure is as follows:

After the first two introductory chapters, a section is dedicated to define the most important nSHIELD system terms.

Then, chapter 4 introduces the adopted design methodology, including the architecture design process, topics under consideration and a basic set of requirements linked to system architecture.

Chapter 5 includes a description of the transitional process from pSHIELD to nSHIELD, highlighting more useful background knowledge than a framework with continuity.

In this document, emphasis is given to chapter 6, which illustrates the attempt to outline an overall architecture scheme and provide the first views on the four layers of nSHIELD stack.

Chapter 7 of interfaces will be analysed in subsequent deliverables, whereas at this point the importance of a clear definition of interfaces is depicted, along with a first attempt of categorization of interfaces and identification of open issues to be resolved.

Similarly chapter 8 including the validation of architecture through the implementation of the four predefined application scenarios is transferred for the next deliverable versions.

A brief conclusive chapter (9) summarizes the findings and status of nSHIELD architectural design.

3 Terms and Definitions

The general terms and definition listed in [1] and [3] apply also here. For completeness and facilitation of reading we restate the most important of them properly adapted to the nSHIELD case.

Embedded system (ES): is a microprocessor based system that is embedded as a subsystem, in a larger system (which ***may or may not be a computer system***) which may include hardware and/or mechanical parts. Embedded Systems are usually designed to perform one or a few dedicated functions often with real-time computing constraints. This is in contrast to a general purpose computer system (i.e. a personal computer) that is designed for openness and flexibility in order to meet a wide range of end-user needs.

Middleware: Middleware is software that has been abstracted out of the application layer for a variety of reasons [10]. The line between middleware and application software is often blurred. Generally, middleware provides services to software applications beyond those available from the operating system. It can be described as a kind of "software glue" [11] that make it easier for software developers to perform communication and input/output, by hiding operational system's details from the application developer, so they can focus on the specific purpose of their application.

According to [10] in an embedded system middleware can be defined as system software that typically sits on either the device drivers or on top of the OS, and can sometimes be incorporated within the OS itself. It acts as a mediator between application software and the kernel or device driver software. But can also mediate and serve different application software. Specifically, middleware can be seen as an abstraction layer generally used on embedded devices with two or more applications in order to provide flexibility, security, portability, connectivity, intercommunication, and/or interoperability mechanisms between applications. One of the main strengths in using middleware is that it allows for the reduction of the complexity of the applications by centralizing software infrastructure that would traditionally be redundantly found in the application layer.

There are many different types of middleware elements, including message oriented middleware (MOM), object request brokers (ORBs), remote procedure calls (RPCs), database/database access, and even networking protocols that run above the device driver layer and below the application layers of the OSI model. They can be categorized as:

- general-purpose, meaning they are typically implemented in a variety of devices, such as networking protocols above the device driver layer and below the application layers of the OSI model, file systems, or some virtual machines such as the JVM.
- market-specific, meaning they are unique to a particular family of embedded systems, such as a digital TV standard-based software that sits on an OS or JVM.

nSHIELD system: an nSHIELD system can be seen as a system composed of a set of interconnected embedded systems that can seamlessly interact through appropriate interfaces that provide at least specific security, privacy and dependability (SPD) functionalities.

nSHIELD Architecture (nSA): describes the overall architecture of an nSHIELD enabled system that is comprised of functional entities (i.e. a software functionality, a middleware service, an abstract object, etc) and physical entities (e.g. a hardware component).

Inheriting from the pSHIELD case nSHIELD defines four main functional layers: *node*, *network*, *middleware* and *overlay*, which represent a set of four functional sub-systems that are specified by a set of elements, functional entities and interfaces:

- **Node Layer**: This layer is composed of standalone and/or connected devices elements like sensors, actuators or more sophisticated ES devices, which may perform smart transmission. Generally it includes the hardware components that constitute the physical part of the nSHIELD system.

- **Network Layer** is a heterogeneous layer composed by a common set of protocols, procedures, algorithms and communication technologies that allow the communication between two or more nodes as well as as with the external world.
- **Middleware Layer** includes the software functionalities that enable:
  - basic services to utilize underlying networks of embedded systems (like service discovery and composition)
  - basic services necessary to guarantee SPD
  - execution of additional tasks assigned to the system (i.e. monitoring functionality)

  Middleware layer software is installed and runs on nSHIELD nodes with high computing power
- **Overlay Layer:** is a logical vertical layer that collects (directly or indirectly) semantic information coming from the Node, Network and Middleware layers and includes the "embedded intelligence" that drives the composition of the nSHIELD components in order to meet the desired SPD level. It is comprised of software routines running at application level.

Figure 3-1 provides a conceptual picture of the four functional layers of nSHIELD system together with a number of important SPD properties that must be considered. As it becomes evident from the figure this properties (i.e. energy) cannot be regarded to be part of a single layer but rather they impose cross-layer constraints and requirements.

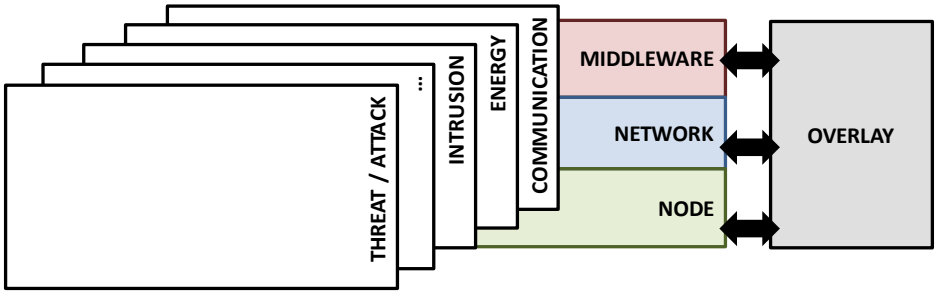

Figure 3-1: the four functional layers of an nSHIELD system

4 Design methodology

The design methodology describes the process that will be followed and the design decisions/considerations made. These design decisions and/or strategies may affect the overall system and its higher-level structures while they provide insight into the key abstractions and mechanisms used in the system architecture.

4.1 Architecture Design Process

The architecture definition activities are based on a slightly modified Embedded Systems Development Lifecycle Model [17] and are depicted in Figure 4-1. Based on the initial system concept defined in the technical annex and the identified basic nSHIELD scenarios a preliminary list of requirements was derived that drove the initial creation of the nSHIELD architecture. The present deliverable (D2.3) details this initial version of this architecture which is also largely influenced from the pSHIELD project experience. A subsequent version of the deliverable will update and refine the nSHIELD architecture based on additional feedback from involved stakeholders and while additional requirements become available from nSHIELD deliverable D2.2 on "Preliminary System Requirements and Specifications" [3]. A Reference Architecture will be provided in D2.4, whereas the final version of the architecture (defined in D2.4) will form the basis for the subsequent system development activities (prototypes) that will be performed in WP3, WP4 and WP5.

As it becomes evident requirements and architecture influence one another. Requirements are the main input for the architectural design process since they explicitly represent the stakeholders' needs and desires and also state the architecture constraints. On the other hand during the architecture design one has to take into considerations what is possible and look at the requirements from a risk/cost perspective.

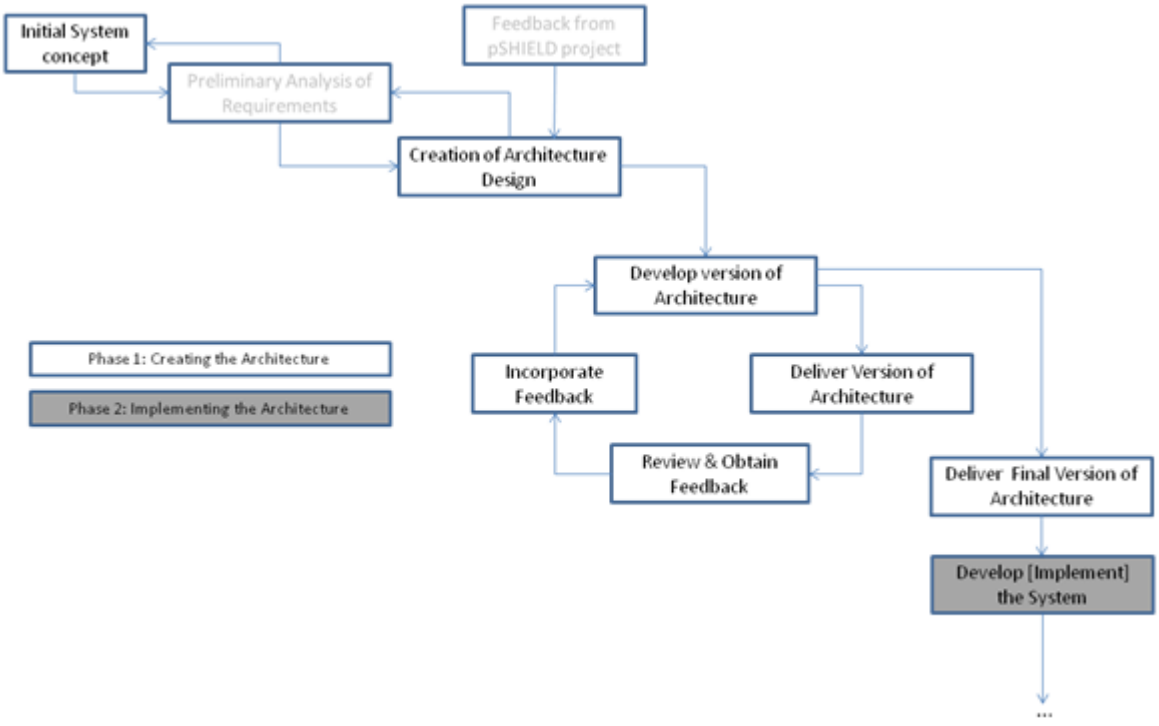

Figure 4-1: process of Architecture definition in the nSHIELD case

For the architecture definition both functional and non-functional requirements (performance, security, reliability, testability etc.) should be examined. In order to define the architecture a top down approach is suggested where initially the entire nSHIELD system is considered and its physical and logical topology is described. After the overall architecture is adequately described we proceed with the analysis of the 4

functional layers starting from the list of provided services and going as deep as to define basic hardware, software or other types of elements that implement these services. The level of details for each functional layer may differ but it is essential that in all cases information is provided at least regarding the available services, their possible interactions and interdependencies as well as the information that flows between the nSHIELD layers.

Regarding the description of the nSHIELD functional layers, the adopted methodology is based on a "viewpoints driven" approach. The principles and recommended practices described in IEEE 1471 [18] and its successor IEEE ISO/IEC 42010:2007 [19] should be followed. Based on these standards a number of different views that can be used to provide a sufficient description of the architecture within each layer are defined. A *view* should be regarded as a representation of the whole system from the perspective of a related set of concerns. Each view is actually governed by one architecture *viewpoint* which is in effect a specification for an architecture view (the view has to conform to its viewpoint).

A number of different architecture frameworks exist in system and software engineering that conform to the "viewpoints driven" approach each one seeking to establish a common practice for creating, interpreting, analysing and using architecture descriptions within particular domains for sufficiently representing a system. Examples of such frameworks include MODAF [20] and DODAF [21], developed initially for describing systems of defence domain, TOGAF [22] that targets enterprise information architectures, 4+1 Architecture View model [24] for software intensive systems, RM-ODP [23] for the standardization of open distributed processing systems and many others.

For nSHIELD a number of 4 views are proposed for describing each one of the 4 functional layers. The proposed views are influenced by the 4+1 Architecture View model which although initially defined to address purely software systems it provides a quite intuitive and well defined approach for describing all nSHIELD layers in a uniform manner.

For modelling the various views related information the Universal Modelling Language (UML) developed by the Object Management Group (OMG) is proposed. UML defines notations and semantics for describing in an intuitive visual manner (UML diagrams) both behavioural and structural elements of a system's architecture.

The views selected to be used for describing the nSHIELD layers are:

- *Logical view*: The logical view is concerned with the functionality that the layer. This diagram should provide an overview of the services and capabilities offered at each layer. In most cases UML class diagram should be used.
- *Development view*: The development view illustrates a system from a programmer's perspective and is concerned with software management. A combination of the UML Component diagram and Package diagram can apply in this case.
- *Process view*: The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behaviour of the system. UML Diagrams to be used include behaviour diagrams like Activity diagram, Sequence diagram or State diagram.
- *Physical view*: The physical view depicts the system from a system engineer's point-of-view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is also known as the deployment view. UML Diagrams used to represent physical view include the Deployment diagram while component diagrams can also be utilized.

Although not all of the above views may be fully applicable to each of the 4 nSHIELD functional layers the use of the architecture structures described above is expected to facilitate overall design by providing a separation of concerns since it is generally pretty difficult to reflect all the information about the system (layer in our case) in only one type of structure. However, it should be denoted that the various structures are different perspectives of the same system, and therefore are not completely independent of each other. This means that at least one element of a structure may be represented as a similar element or some different manifestation in another structure, and it is the sum of all of these structures that makes up the architecture of the nSHIELD system.

4.2 Design Considerations

nSHIELD continues and improves the results of pSHIELD project [1]. The overall SHIELD concept aims at addressing Security, Privacy and Dependability (SPD) in the context of Embedded Systems (ESs) as “built in” rather than as “add-on” functionalities, proposing and perceiving with this strategy the first step towards SPD certification for future ES. Therefore an nSHIELD System can be described as a set of interacting and interconnected embedded systems each one providing specific composability capabilities and SPD Functionalities. In the following subsections we will briefly set out a number of considerations that should be taken into account and might drive some important selections during nSHIELD system architecture definition.

4.2.1 Distributed vs. Centralized Approach

An nSHIELD system is comprised of heterogeneous devices combining several hardware and software components such as content, applications, displays, etc. Such devices will share their content with other users usually over a network. The nSHIELD project aims at providing a framework that ensures security, privacy and dependability for applications that may execute over a highly distributable network of embedded nodes. The distribution can be considered to occur on two different levels: on a conceptual level where information is distributed and on an implementation level where system components are distributed. In the latter case, the management of distributed components can occur in a centralized or decentralized manner.

A centralized approach is based upon a centralized component or server for several types of information and services, which provide requested information to the applications running on several devices. This approach decouples the acquisition of information (content, user-related information, context, device properties, etc.) from the processing of this information. These applications can actively request the desired information from the server or passively be notified about changes. The server collects all information from accordant acquisition components and provides it to interested applications. A centralized approach suffers from restricted scalability while the problem of privacy rises, since all user-related information is bundled and stored in one place.

Instead of maintaining all information and services in one centralized place, a distributed approach holds the information at several places to avoid potential bottlenecks. Small devices maintain the information required by the application and process it directly. This approach requires the device to have the capability to store and process all of the necessary data, which may not be efficiently achieved for a simple device with restrictions concerning space, weight, or energy consumption. The decentralized approach circumvents the lacking scalability of the centralized approach and allows finer control on the way device information is published and protected. On the other hand, the sharing of information while preserving privacy is an issue.

In nSHIELD due to diversity and nature of applications that need to be supported the distributed approach seems preferable. This will ensure scalability while the use of appropriate middleware and overlay services will ensure a service architecture that encapsulate the complexities and privacy issues that may arose.

4.2.2 Service oriented architecture

In general embedded devices/systems tend to be secluded and isolated without providing easy ways to add a custom interface to them. This was due to the fact that they are regarded as processor based systems designed to provide dedicated tasks in comparison to “big” computer systems (i.e. servers) that are designed to be open and extensible. In addition to that, embedded devices usually have certain limitations in terms of resources like memory, power and communication infrastructure. The current trends of convergence of computing and communication have partly addressed some of these limitations by for example making embedded systems capable of communicating using different wired and/or wireless technologies. However, embedded systems are still mainly seen as vendor-specific and task-oriented products, and not as components that can be easily manipulated and reused. Therefore, considerable steps are further needed in order to make embedded systems more accessible.

Borrowing ideas and concepts from software systems design, the architecture approach adopted for nSHIELD proposes to apply, as a first step, a ***component-based paradigm*** in nSHIELD where embedded devices are initially regarded as components with strictly defined interfaces. In a second step, the architecture should also adopt ***service-oriented computing*** principles. In service-oriented computing, services are basic building blocks for application development. Services are self-describing and open components that support rapid and seamless access and integration. Services are offered by service providers and they are used by service consumers. Therefore, the main architectural units in service-oriented computing are *service description*, *service discovery* and *service consummation*. This means that the potential nSHIELD architecture should consider at least:

- Ways to describe the services (required or provided)
- Ways to discover them
- Means and infrastructure to enable their usage

Addressing the above will require modifications/enhancements at the node level since embedded systems need to provide additional functionalities (i.e. in order to be discovered or composed). These functionalities can be implemented in most cases as an extra software add-on module that runs on the embedded node. An important requirement though is to minimize the needed changes in incumbent systems employing legacy nodes, without compromising their ability to be part of a future nSHIELD system. This is not a simple task since legacy embedded nodes may have limitations, such as lack of operating system or of enough memory, that does not allow the deployment of even a minimal set of additional software capabilities. As it will become evident in chapter 0 (section 6.1), the proposed architecture has addressed that by introducing the notion of an nSHIELD subsystem which prescribes a cluster-like architecture. This enforces that at least one embedded system node with advanced SPD functionalities must be present in each cluster. This node can be configured to act as *proxy* or provide *adapter* functionality for the rest devices that do not have the ability to directly expose enhanced functionalities.

4.2.3 Middleware considerations

The service-oriented architecture prescribes the need of an appropriate middleware, a kind of framework that will support service description, discovery and consummation. Indeed nSHIELD systems can include distributed devices and therefore it is essential to have middleware that makes it easier to write distributed applications and takes care of all the networking code and messaging required. Today there are many service architectures proposed. The most prominent of them are Web Service Architecture [7] and the OSGi Service Platform [4], while there are other approaches like JINI services [6] or Open Grid Service Architecture (OGSA) [5]. More traditional approaches include the Common Object Request Broker Architecture (CORBA) [12] and Microsoft's proprietary Distributed Component Object Model (DCOM) [13]. Although the majority of them were initially developed to address general purpose computer systems, their basic principles could possible apply in embedded systems too. A thorough study is needed in order to identify what kind of description is needed for embedded systems and how to modify the way embedded systems are designed in order to be able to access and use them in a service-like architecture.

4.2.4 Evolving from interfaces to contracts

The first task that needs to be solved, in order to provide a service-oriented architecture for embedded systems, is to propose a way for uniform specification of systems that would constitute such architecture. Although, it is a great challenge to come up with a specification scheme that is both general and lightweight, it is impossible to provide a meaningful interface specification of an open component without considering the context-of-use in a particular application environment. Therefore, the issues that are conceptually important when trying to specify a component, be it a general purpose software component, a web server, or an embedded system will be initially marked. After the basic issues are covered the peculiarities of embedded systems specification will be considered.

A component's interface usually describes the functionality exposed by a component (provided interface). In a more general case the interface can also include information on behaviour requested by the component (required interface). In theory this information will be enough for a client to decide how to use a component. However knowing how to use a component is only one thing. In safety-critical systems, as

is the case for a great portion of embedded systems, a client should be aware what and how a component will deliver, too. This means that *a component's interface must be augmented with additional information that relates to non-functional properties such as security, dependability, performance* etc. Such an extended interface can be regarded as a kind of contract. There is an elaborate effort to introduce contracts into modern software engineering [8] [9]. Contracts are considered a good prospect to have when building a service-oriented architecture since:

- Equipping components with contracts facilitates reuse and makes it much safer.
- Contracts can help in comparing and choosing between similar components.
- Contracts fit perfectly as semantically extended service descriptions, which allow treating components as services.
- Adding composition behaviour to contracts can help with automated component composition.

One of the major problems of developing contracts for embedded systems is the fact that "embedded system" does not cover just one concept or class of devices. Instead, "embedded systems" means a whole range of devices from very small low power singles solutions up to large multiprocessor systems. Compared to commercial-off-the-shelve (COTS) computers, embedded systems are typically characterised by limitations in resources such as CPU cycles, storage, power and software.

4.2.5 Interconnectivity of embedded devices

In terms of interconnectivity it is not possible to assume that each embedded system is able to communicate using some standard communication protocols because in general this is not needed to fulfil the needs of the application. Therefore, new communication schemes such as service oriented architectures have to obey that fact and should not try to force the usage of a specific high-level protocol for each device. On the other hand, such architecture would make no sense without the ability to interact "somehow" with all kinds of devices. Considering embedded systems in terms of communication capabilities we can classify them to the following four (4) categories [14]:

- **No communication capabilities (n-ESD)**: In very limited application domains devices are used that only interact with their physical environment and have no possibility to exchange information with other devices. Such a device is completely isolated and cannot be included into any kind of communication architecture therefore we will not consider it during architecture specification.
- **Proprietary physical communication capabilities (pp-ESD)**: these devices (which are the majority today) are systems that are able to interchange information using proprietary methods both at physical and logical layer. This ability does not necessary mean that the device is "networked", it is sufficient that it is able to deliver data to other systems. Examples here are control systems in cars for, e.g., airbags, engine, comfort functions or the braking system.
- **Proprietary logical communication capabilities (pl-ESD)**: With increasing complexity devices may use standard communication techniques at the physical level (e.g., Ethernet) or both at physical and transport level (e.g., Ethernet and IP) and a proprietary protocol above that to interchange data with other systems that use the same technology. Example here is, e.g., the remote control of some cameras using Ethernet links.
- **Full communication capabilities (f-ESD)**: includes embedded systems that implement the full communication architecture and are able to interact with all other systems of the architecture.

In nSHIELD we should consider embedded nodes that may belong to any of the latter three classes and the envisaged service architecture should cater for providing the necessary middleware and network layer functionality to address all three cases. The architecture should promote a platform-independent communication mechanism to the extent that this is possible. Existing approaches to provide service oriented connectivity to embedded devices with heterogeneous characteristics include the Hydra middleware [15] and Prism-MW [16]. The former offers an easy-to-use web service interfaces for controlling any type of physical device irrespective of its network technology such as Bluetooth, RF, ZigBee, RFID, WiFi, etc. Moreover, Hydra incorporates means for Device and Service Discovery, Semantic Model Driven Architecture, P2P communication, and Diagnostics while it provides distributed

security and social trust components that can ensure security and trustworthiness of Hydra enabled devices and services. Prism-MW on the other hand is described as middleware targeted at applications in highly distributed, resource constrained, heterogeneous, and mobile settings. Its key properties are its native, and flexible, support for architectural abstractions (including architectural styles), efficiency, scalability, and extensibility. Implementations exist in both C and Java programming languages.

4.3 Requirements on Architecture

The methodology adopted (D2.2) identifies two major categories, namely Functional and SPD Requirements. Furthermore, these two groups are internally discriminated according to Scenarios (4 Application Domains) and in a more low level according to Layers (4 nSHIELD Layers). From the wide set of requirements identified in nSHIELD (an on-going procedure), a registration of the most influential/related to system architecture follows hereafter. This is just a first incomplete refinement of previously mentioned and new attributes, trying to capture the rationale behind the interactive association of requirements and architecture.

Table 4-1: Requirements

REQUIREMENT ID	DESCRIPTION	MODULE/ SUBSYSTEM
REQ_D2.1.1_0501.A	The SHIELD middleware shall offer discovery functionalities	Middleware
REQ_D2.1.1_0502.A	The SHIELD middleware shall be able to compose SHIELD components	Middleware
REQ_D2.1.1_0503.A	The SHIELD middleware shall be able to orchestrate, according to defined policies, the composition of SHIELD components	Middleware
REQ_D2.1.1_0504.A	The SHIELD middleware shall be able to retrieve information from SHIELD components	Middleware
REQ_D2.1.1_0505.A	The SHIELD middleware shall be able to enforce decisions into SHIELD components	Middleware
REQ_D2.1.1_0506.A	The SHIELD middleware shall be able to interface with heterogeneous legacy component	Middleware
REQ_D2.1.1_0507.A	The SHIELD overlay shall be able to elaborate feasible system configurations	Overlay
REQ_D2.1.1_0508.A	The SHIELD middleware shall implement secure Discovery	Middleware SPD
REQ_D2.1.1_0509.A	The SHIELD middleware shall implement trusted Composition	Middleware SPD
REQ_D2.1.1_0510.A	The SHIELD middleware shall verify core services integrity	Middleware SPD
REQ_D2.1.1_0210.A	An nSHIELD node should provide *vitality checking* capabilities	Node
REQ_D2.3_0100.A	An nSHIELD node should provide *cryptographic transmission* capabilities	Node
REQ_D2.3_0200.A	nSHIELD should guarantee *message sequencing* between nodes	Network/ Middleware
REQ_D2.3_0201.A	nSHIELD should guarantee *message delivery* between nodes	Network/ Middleware
REQ_D2.3_0202.A	nSHIELD should guarantee *message integrity* between nodes	Network/ Middleware
REQ_D2.3_0203.A	nSHIELD should provide mechanisms for *time constrained data delivery* between nodes	Network/ Middleware

REQ_D2.1.1_21120.A	nSHIELD should provide capabilities for checking *authenticity* of received data	Network/ Middleware/ Overlay
REQ_D2.3_0300.A	nSHIELD should provide capabilities for *authorized* access to a node	Network /Middleware /Application
REQ_D2.3_0204.A	nSHIELD shall be able to support heterogeneous transmission technologies	Network
REQ_D2.3_0101.A	nSHIELD nodes shall be able to implement TPM modules	Node
REQ_D2.3_0102.A	nSHIELD low cost nodes shall implement asymmetric cryptography	Node
REQ_D2.3_0103.A	nSHIELD nodes shall implement lightweight HW and SW crypto technologies	Node
REQ_D2.3_0205.A	Cryptographic protocols should guarantee data integrity	Network/Node
REQ_D2.3_0104.A	nSHIELD nodes should be able to perform self-re-configurability and self-recovery of sensing and processing tasks (for no energy-constrained nodes)	Node

5 From pSHIELD to nSHIELD

Some of the small milestones towards the definition of the nSHIELD System Architecture design are summarized at the following:

- Exploring interdependencies between applications and architectures
- Including critical elements and covering SPD application requirements
- Developing the 4 functional layers, composed from HW and SW modules
- Taking into account reconfigurability, tailoring overall system needs
- Defining interfaces, interconnecting different SPD modules
- Connecting layers, ensuring secure routing of information
- Producing a composable architecture, that meets the requirements of desirable SPD levels

pSHIELD acted as a feasibility study and in some terms was used to propose, test or implement a subset of the expected SHIELD structures and functionalities. The composability of foreseen SHIELD technologies was investigated only to design level. A basic set of preliminary metrics was used to validate the first basic functionalities. A bunch of use cases was presented, loosely connected and with different levels of implementation, to evaluate, at a first stage, the suggested architectural framework. These paradigms mainly stemmed from a single application scenario. The core of a high dependable reference Architecture was designed, incorporating innovative, modular and composable elements, leaving to nSHIELD its refinement and further development.

The outcome of this effort is based on a conceptual architectural model, synthesized incrementally from newly introduced components. Three different types of Embedded Devices were proposed, with hierarchically increased processing capabilities. The notion of a pSHIELD Subsystem was introduced. Combinations of subsystems constitute a pSHIELD System Architectural scheme. Additionally, a set of internal modules, functionally critical for the system, were defined (e.g. pSHIELD Proxy, pSHIELD Adapter and Security Agent). Detailed descriptions can be found on pSHIELD deliverable D2.3.2 “System Architecture Design”.

In nSHIELD we plan to manage pSHIELD background as a useful but non-committing substructure. A wider set of technologies will be used to realise SPD composability. Metrics will be updated and expanded, whereas work plan foresees the evaluation of system’s performance through four complex application scenarios. Requirements and Architecture will be placed in close interaction. Interfaces used by nSHIELD components to interact with neighbouring or remote world shall be clearly defined. The objective is to result in the design of the nSHIELD Architecture that will provide or encompass:

- Requirements, Metrics, Scenarios, Technology Status and Advancement
- Functionalities successfully addressing Security, Privacy and Dependability (SPD) in the context of Embedded Systems (ESs)
- A functional model of 4 layers
    - Node: intelligent hardware and firmware SPD
    - Network: trusted data transfer
    - Middleware: resource management, service discovery and network interoperation
    - Overlay: composability orchestrated by the Security Agent modules
- Logical and physical interfaces facilitating the internal and external communication and overall system effectiveness

6 nSHIELD Architecture

This chapter focuses on providing detailed information regarding the nSHIELD system architecture. The nSHIELD architecture is considered and analysed following the design consideration and overall strategy described in section 0. Two different aspects are considered. Initially, in section 6.1, an overview of the overall nSHIELD system, seen as a network of interconnected embedded devices, is provided trying to elaborate on the various types of ESDs that are supported and the hierarchical nature of the network. Sections 6.2 to 6.5 address individually the 4 functional layers comprising nSHIELD.

Note that for this first version of the nSHIELD architecture document, only the logical view is available for all functional layers. This view provides mainly information regarding the major services and capabilities supported by each layer. While activities on requirements analysis and architecture specification are still in progress, additional views will become available in future versions of the architecture definition document (nSHIELD deliverables D2.4 and D2.7 according to the Technical Annex). For some layers additional views do exist (i.e. a development view does exist for the network layer).

6.1 nSHIELD Overall Architecture

A nSHIELD system is expected to primarily consist of medium to high power embedded devices that are equipped with all the needed SPD functionalities and can seamlessly interact through appropriate service oriented interfaces.

However, as already mentioned in section 4.2 (design considerations) an nSHIELD enabled system may include legacy embedded devices (L-ESD) with:

- significant resources constraints, such as lack of operating system or of enough memory that does not allow the deployment of even a minimal set of additional software (SPD) capabilities
- proprietary physical (pp-ESD) or logical (pl-ESD) communication capabilities that do not allow direct interconnectivity to a service oriented architecture (essentially do not directly support the nSHIELD network protocols or middleware services)

In order to address that, the proposed architecture introduces 3 additional types of embedded devices that are:

- ***nSHIELD Embedded System Device*** (**nS-ESD**): This is the basic element of the nSHIELD network. It implements the minimum SPD capabilities that relate to the 3 first layers of the nSHIELD functional architecture (node, network and middleware).
- ***nSHIELD Embedded System Device Gateway*** (**nS-ESD GW**): In terms of SPD capabilities this type of device could be regarded as identical to the nS-ESD. However, it may provide some enhanced capabilities in terms of interconnectivity that will allow L-ESDs of type pp-ESD and pl-ESD to overcome communications issues and interact through the nSHIELD middleware[1]. These devices may exist at the border between an nSHIELD network of devices and a network of Legacy embedded systems.

[1] This type of adapter functionality may be described as:

- *Translator* like behaviour: Intercepts service requests and transforms them into a logical format that an L-ESD can understand. Physical conversion is not needed since the L-ESD uses a standard physical communication medium. Therefore transformation concerns mainly translation of service requests at middleware layer and above
- *Proxy* like behaviour: Transforms service requests into the physical and logical format that an L-ESD can understand and vice versa. This mainly provides support for legacy devices with non-standard physical communications. Therefore transformation concerns also network layer

- ***nSHIELD SPD Embedded System Device*** (**nS-SPD-ESD**): This is a node which provides a full implementation of the core services required by the overlay layer. Essentially this node does not need to be an embedded device since overlay services are application level and therefore a general purpose computing unit can host them. However, in order to promote a more clear design we will consider **nS-SPD-ESD** as an enhanced nS-ESD embedded platform that hosts also a security agent component (a software module that provides the necessary overlay interface and will be analysed in details in section 6.5).

Therefore nSHIELD can be regarded as a network consisting of nSHIELD and Legacy embedded devices having a physical architecture similar to the one depicted in Figure 6-1. The L-ESDs since they do not understand nSHIELD middleware services they need a gateway nSHIELD device in order to participate in the nSHIELD system.

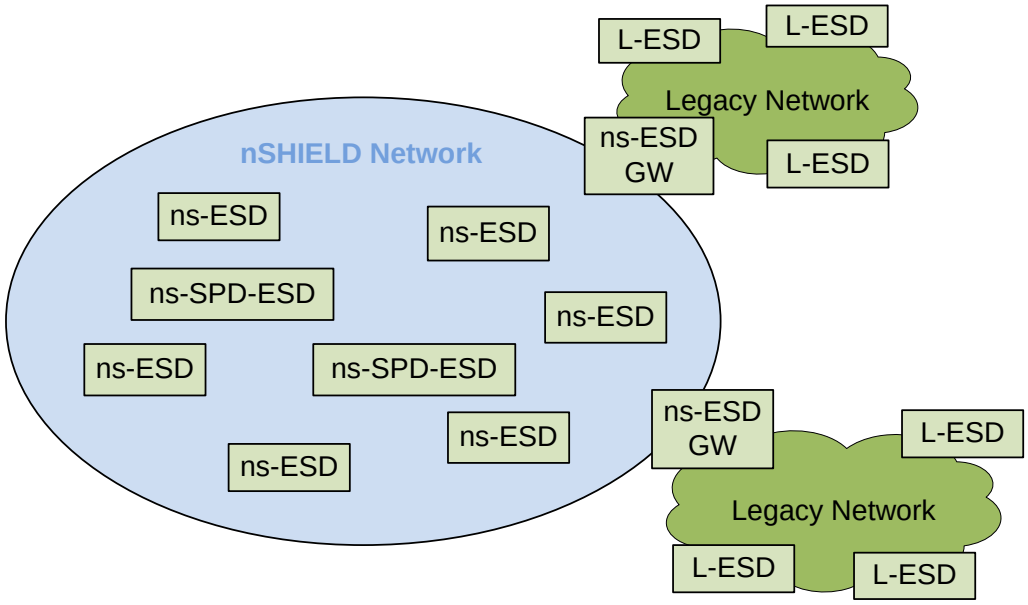

Figure 6-1: Conceptual Architecture of the nSHIELD system (Physical view)

Through Figure 6-1 it is difficult to visualize and understand many of the nSHIELD concepts related to the 4 defined functional layers. Therefore in the following paragraphs we will try to provide some additional information and diagrams that will make some of nSHIELD aspects more evident.

First of all we define the concept of an *nSHIELD aware cluster* (see Figure 6-2). This is a set of ESDs that includes (at a minimum) one embedded node with at least **nS-ESD GW** capabilities. The **nS-ESD GW** provides, to the L-ESDs, the appropriate functionality to interface with the nSHIELD middleware. External to the cluster ***nSHIELD*** devices are not aware of the internal cluster structure unless this has been provided as part of a requested service (at middleware or overlay layers). In general the **nS-ESD GW** at this level is responsible for providing proxy or adapter services for L-ESDs. The "Legacy Network or Middleware" cloud abstracts the physical and/or logical communication capabilities between the nS-ESD GW and the various L-ESDs.

An *nSHIELD subsystem* may contain one or more **nS-ESD** (or *nSHIELD aware clusters*) and at least[2] one **nS-SPD-ESD** capable node that will provide support for nSHIELD overlay services to the underlying

[2] Actually the normal case is that only one nS-SPD-ESD device should be present at the *nSHIELD subsystem* level. This is due to the fact that the nS-SPD-ESD hosts the Security Agent component which is responsible for monitoring, gathering metadata and generally controlling all underline nSHIELD subsystem's devices. The Security Agent must be uniquely identified by all nS_ESD nodes that provide information to it. Therefore, the presence of more than one SPD Security Agent might cause problems and is only justified by the need of solving scalability or availability issues.

elements. The conceptual architecture of an nSHIELD subsystem is depicted in Figure 6-2. The "nSHIELD Middleware/Network" cloud abstract all the network infrastructure and basic services supported by the nSHIELD middleware.

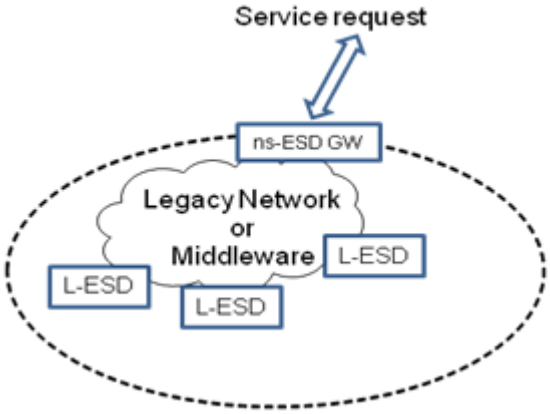

Figure 6-2: Architecture of an nSHIELD aware cluster

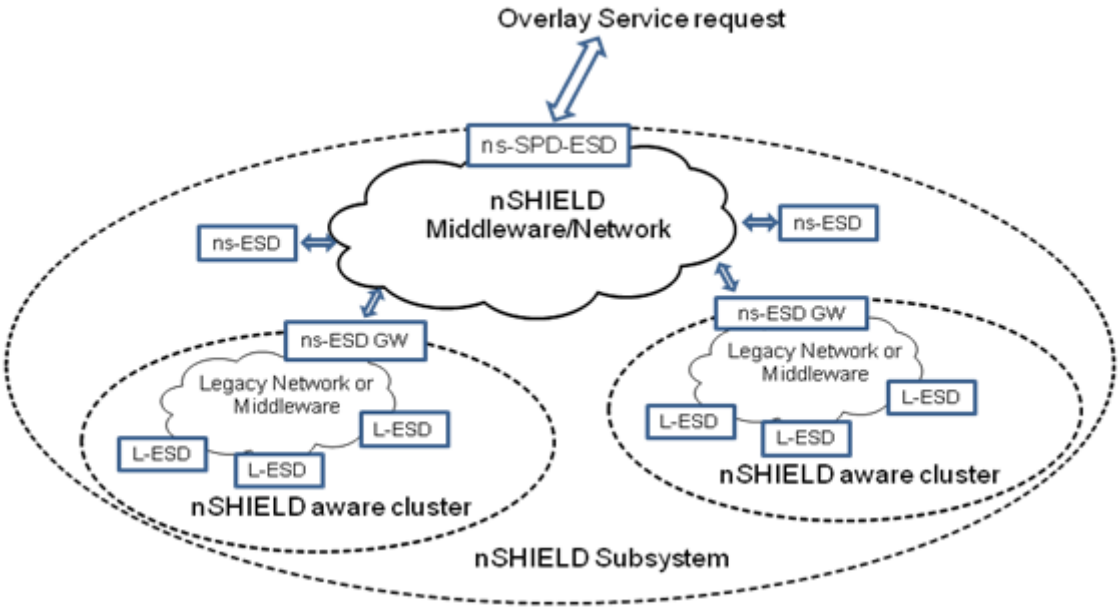

Figure 6-3: Architecture of an nSHIELD subsystem

Putting everything together the overall conceptual architecture of an nSHIELD system is illustrated in Figure 6-4 . The "nSHIELD Overlay" cloud abstract all the SPD and other capabilities as defined for the overlay layer. Figure 6-4 demonstrates the hierarchical structure of an nSHIELD network of devices consisting actually of 3 tiers. If we consider a top down approach these tiers can be described as the nSHIELD System (the overall picture including everything), the nSHIELD Subsystem (a group of ns-ESD devices that are controlled by a nS-SPD-ESD node) and the nSHIELD aware Cluster (a group of L-ESD devices controlled by a nS-ESD GW node)

The figure should be regarded more as a logical architecture for the nSHIELD system. Although the nodes correspond to the physical embedded devices that exist in the system, the various clouds attempt to abstract the interfaces and the way devices exchange SPD related information. They provide a hint on the functional layers involved when the various types of nSHIELD nodes communicate. The ns-SPD-ESD devices interact through the nSHIELD overlay interface while the ns-ESD (GW) nodes interact via the

provided nSHIELD middleware services. The nSHIELD network borderline is also depicted in order to ease the association with Figure 6-1.

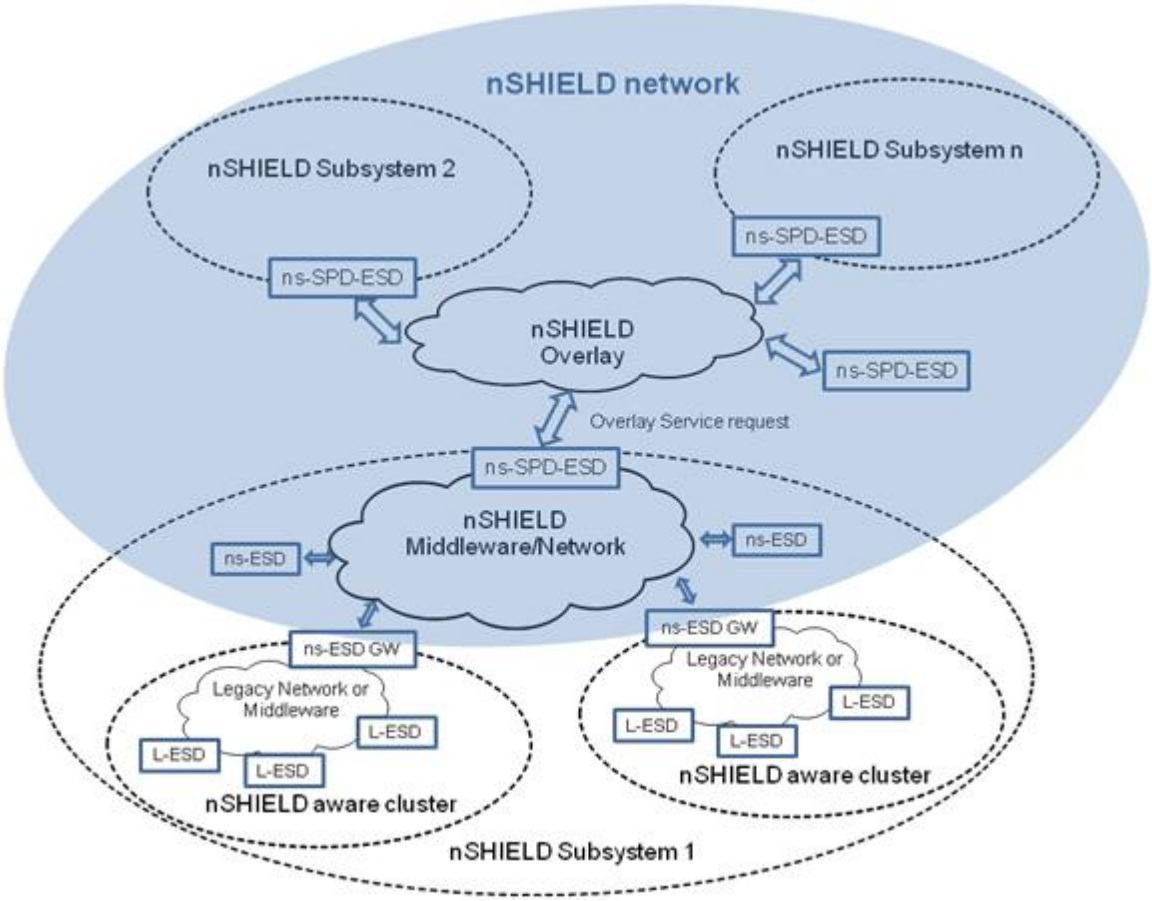

Figure 6-4: Conceptual Architecture of nSHIELD System (Hierarchical logical view)

It must be noted that the types of nodes described above consist a logical categorization. In terms of node capabilities the nSHIELD nodes can be discriminated to:

1. *Nano nodes*
2. *Micro/personal nodes*
3. *Power nodes*

Nano nodes are typically small ESD with limited hardware and software resources, such as wireless sensors. *Micro/Personal nodes* are richer in terms of hardware and software resources, network access capabilities, mobility, interfaces, sensing capabilities, etc. *Power nodes* offer high performance computing in one self-contained board offering data storage, networking, memory and multi-processing. These three nSHIELD node types, which are also prescribed in the Technical Annex, cover a variety of different ESDs, offering different functionalities and SPD capabilities. While an nS-ESD can map to either a nano, micro or power node, the nS-SPD-ESD type should preferably be implemented as a power node since overlay services may require some significant computing capabilities including the ability to process multiple requests.

Having defined the overall architecture and following a top down analysis approach the next step is to refine the internal architecture of the nS-ESD (GW) and of the nS-SPD-ESD. This is done in Figure 6-5 where apart from the functional layers, some coarse grained information on the nSHIELD related data and control flows is also depicted. More fine grained information on these flows will be provided in section 0 which focuses on the interfaces.

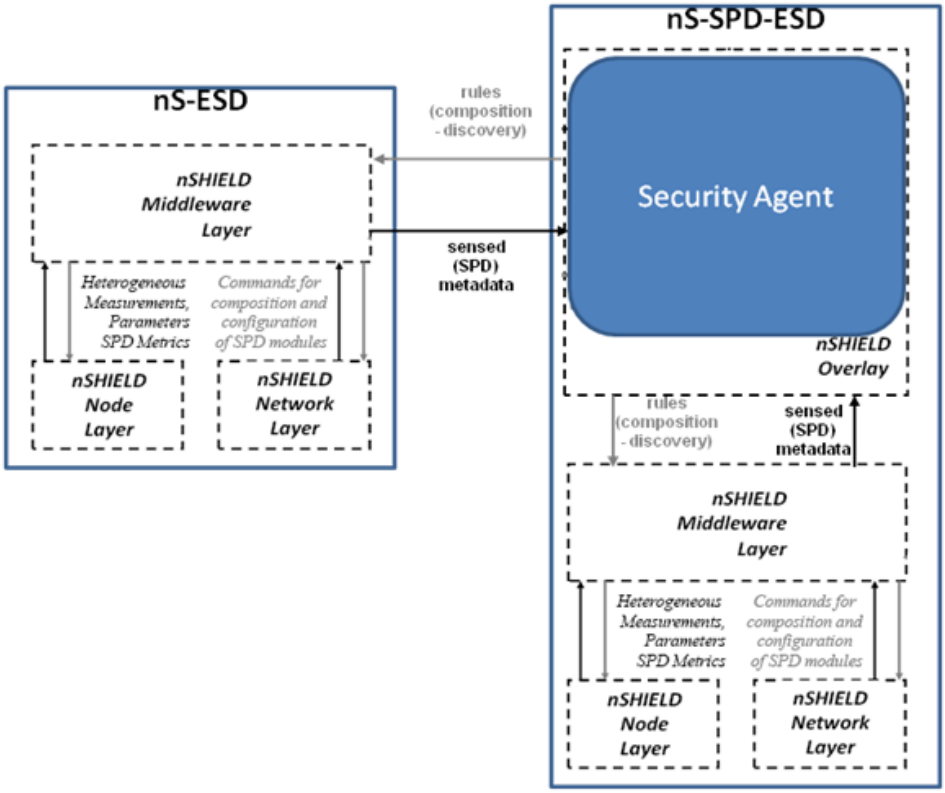

Figure 6-5: Internal architecture of nSHIELD ESDs with respect to the 4 functional layers

6.2 Node

As defined in the TA, the node layer should provide SPD intrinsic capabilities through the creation of an intelligent hardware and software platform consisting of different kinds of intelligent ES nodes. The main SPD features are summarized in the following table:

Table 6-1: SPD features [node layer]

Security	• TPM and Smartcard • Lightweight HW and SW crypto technologies • Asymmetric cryptography for low cost nodes • Intrinsically secure ES firmware
Privacy	• Automatic Access Control • Data compression techniques • Lightweight HW and SW crypto technologies • Asymmetric cryptography for low cost nodes
Dependability	• Power Supply Protection • Self-re-configurability and self-recovery of sensing and processing tasks • Easy and Dependable interfaces with sensors • Embedded camera array auto-calibration and auto configuration techniques

In previous deliverable of this task [3], the main requirements for each layer have been defined and described. In this section based on the functional and SPD requirements, a description of the expected node layers services will take place followed by appropriate diagrams that will provide further insight on the hardware (and software) architecture that should be adopted in order to support the specified capabilities.

6.2.1 Logical View and Services Description

Based on Table 6-1, a list of services, which are implemented at node level, can be identified. Figure 6-6 provides a logical view of all possible services while more details on them is provided in the next paragraphs.

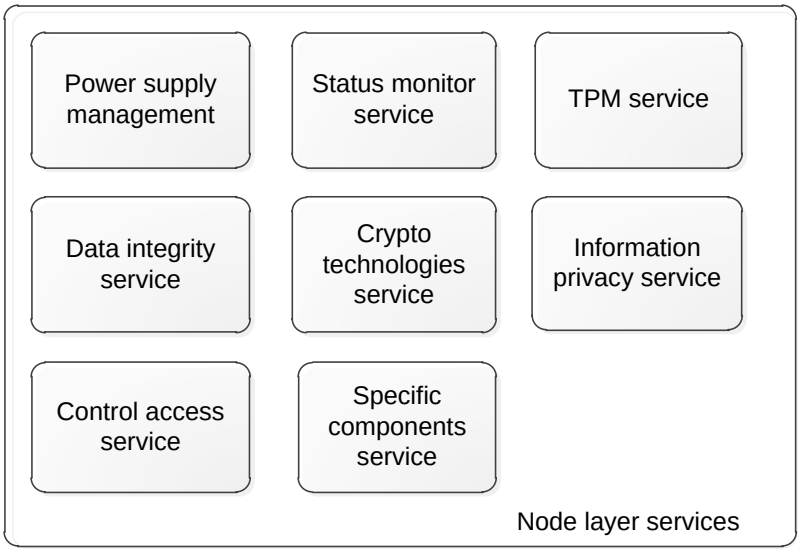

Figure 6-6: nSHIELD's Node Layer Services

6.2.1.1 Power Supply Management

This module should be design for managing power sources, providing protection against blackouts, etc.

The power supply module of the nSHIELD node should:

- Be able to provide a continuous power supply source, without any cut in time neither in the power, voltage or current levels, to correctly bias the devices
- Monitor and prevent any system power supply risk, which might affect to the node behaviour
- In case of failure of any of the countermeasures, being able to protect all the electronics and devices, in order to avoid further damages into the system

The nSHIELD the NMP-SPD nodes should have power-supply circuits with security and dependability features. Depends on the capabilities of the nSHIELD node, this should:

- Support alternative power modes, depending on the specific application and environmental conditions (e.g. vibration generator, micro-solar cells).
- Be self-powered.
- Support mechanisms to protect itself from any power supply failure.
- Be able to support remote powering, at least to some modules of the device, allowing some functionality to become operational in case of power failure.
- Have provisions for future alternative power sources including super-capacitors and wireless power schemes.

6.2.1.2 Status Monitor Service

This module will be the responsible for collecting the status of each individual component in the node, and providing SPD-relevant parameters and measurements to upper layers. It also checks on system health status for self-recovery, self-reconfiguration and self-adaptation.

A nSHIELD node should monitor its performance parameters and report alert or alarm conditions to the external systems when the defined thresholds are overrun.

Other responsibilities of this module should be to control the fails situations, meaning that node failures are, to an acceptable extent, halting and signalled. Also, a nSHIELD node should have the capability of performing a complete self-test of all its functions.

An nSHIELD node shall be able to provide situational-aware and context-aware SPD services.

6.2.1.3 TPM Service

The technology around the TPM has been developed for few years now through the Trusted Computing Group (TCG)'s initiative. Initially driven by its application for the PC platforms, the component could provide interesting functionalities to a large panel of devices and in particular to embedded systems. For nSHIELD nodes, the TPM module should be extended mainly in order to have inexpensive implementation to allow widespread use and also should be extended in order to implement additional mechanisms to improve product endurance and increase product lifespan.

Depend on the node capabilities this module could include more capabilities like:

- Utilize the TPM remote attestation functionality to ensure the integrity of a node prior to resource allocation.
- Be compliant with global export control regulations in order not to restrict international trade with TC platforms (PCs).
- Implement additional cryptographic protocols (e.g. elliptic curves)
- Extend TPM key generation functionality to include key generators and key parameters that depend on the context available.
- Add some specialized/dedicated commands (e.g. to further develop on-the-fly encryption)
- Have alternative communication interfaces, better adapted to the embedded applications that the LPC (low pin count) currently supported.
- Also should be recommended an improved global architecture of the embedded SW of the TPM to support future evolution of cryptographic/hash
  Be able to enter into a low power state without compromising its security

6.2.1.4 Data integrity

Since the node is the basic component of the nSHIELD architecture, security issues in firmware will be explored as well as the techniques to make it intrinsically secure. Some points to take care about are:

- Code execution
- Intrinsically secure ES firmware
- Data Freshness
- Secure firmware upgrade
- Secure boot
- Protection against Side-Channel Attacks (SCA)
- Physical/tamper resilience

6.2.1.5 Control Access Service

Access control and denial of service mechanisms are in charge of preventing non authorized/malicious entities to access the physical resources of the ES nodes that can be reached over the network.

There are several ways to implement access control in a network, depending on the “intelligence” of the nodes, the memory capabilities and the predefined profiles. Those methods are based on:

1. Profile authentication: If the node has some characteristics, it can join to the network.
2. Access Code (programmable or configurable): Typical password access, based on memory data, switch configuration, or any other procedure
3. Predefined topology: Only pre-established nodes can join to the network, like MAC filtering in a Wi-Fi

The node layer of the nSHIELD architecture should provide basic access control mechanisms to the higher layers and support secure authentication protocols.

Depend on the capabilities of the nSHIELD node, some other features should be offered by this module:

- Verification of digital signatures even in cases where a trusted third party is not available.
- Allow security context establishment and sharing, allowing more efficient keys or key material to be exchanged, thereby increasing the overall performance and security of the subsequent communications.
- Design mechanisms that improve its resilience to unauthorized information alteration (integrity).
- Design mechanisms that improve its availability for authorized users.
- Provide mechanisms that allow secure upgrading of the firmware from a remote site as well as local site.

6.2.1.6 Crypto Technologies Service

At node level cryptographic operations are expected to be performed by low-energy low-processing devices. The SW embedded on such a cryptographic component has a direct impact on its size, its costs, its speed as well as its power consumption.

The nSHIELD Node layer should support lightweight HW and SW crypto technologies. The term lightweight crypto refers to algorithmic designs and implementations best suited to constrained devices. The nSHIELD Node layer should support asymmetric cryptography for low cost nodes and also should include an optimized hardware implementation for an ECC-based public-key authentication algorithm.

In the nSHIELD the Elliptic Curve Cryptography should be implemented on energy constrained NMP-SPD nodes.

This module, depends on the node computation capabilities, also should offer a key management options:

- nSHIELD node shall offer key size parameterisation options that map the requirements of the specific application/scenario, based on the need for short, medium or long-term security.
- nSHIELD node shall support a secure low-cost key distribution mechanism. The mechanism's parameters (e.g. algorithm, key length, usage, entropy etc.) will be defined by the security policy requirements, taking into consideration any restrictions that participating parties impose.
- nSHIELD node may offer support for third-party key management services to compensate for the shortage of ES computational power in constrained
- nSHIELD node shall support secure and dependable low-cost key distribution mechanisms for initialisation or re-keying.

6.2.1.7 Information Privacy Service

Nowadays, wearable and ubiquitous computing is emerging for embedding different kinds of sensory devices on the user’s body or on various items in the environment. This activity requires means to collect, store and label the data wirelessly and in a non-obtrusive way. Taking account the associated privacy risks, a nSHIELD node should generate cryptographic keys using the context available and feature privacy-aware management of location and other sensitive personal information, utilizing secure storage and sanitization mechanisms to be applied to such information prior to transmission.

Another point in this subject is the continued advances in mobile networks and positioning technologies that have created a strong market push for location-based applications. Examples include location-aware

emergency response, location-based advertisement, and location-based entertainment. An important challenge in wide deployment of location-based services (LBSs) is the privacy-aware management of location information, providing safeguards for location privacy of mobile clients against vulnerabilities for abuse. Advances in sensing and tracking technology enable location-based applications but they also create significant privacy risks.

Taking into account these risks, nSHIELD node should feature the necessary mechanisms for security token exchange to enable the issuance and dissemination of credentials within different domains. Also a nSHIELD node should incorporate provisions that ensure the long term storage of private information, not allowing the confidentiality of that information to be compromised even under fault conditions.

Finally a nSHIELD node should feature privacy-aware management of location, utilizing secure storage and sanitization of such information prior to transmission.

6.2.1.8 Specific Components Service

This module is a custom module that should be designed depending on the specific requirements of the legacy nodes connections/interfaces. nSHIELD node internal interfaces and algorithms should be developed and communication protocols optimized in order to extend the minimal service period. The nSHIELD node should be disposed as an integral part of the smart sensors/actuators and therefore the communication needs no encryption. One example could be the camera array configuration. It is planned a study and implementation of a efficient camera autocalibration, thresholding and blob detection and tracking techniques.

6.3 Network

The **network layer** of the nSHIELD 4 layer architecture model includes a number of capabilities and services that relate mainly to:

- Trusted and dependable connectivity
- Smart SPD transmission
- Other Network services

The **network layer** functionalities according to what is prescribed in the nSHIELD TA does not address only functionalities related to layer 3 (network layer) of the OSI reference model like i.e. routing. In the architecture definition we should consider at network level functionalities and capabilities that relate also to physical layer transmission, security and integrity of transmitted data, device identity management, etc. that is broader communication layer capabilities. Therefore it would be better to describe it as Network/Communication layer.

6.3.1 Logical View and Services Description

Figure 6-7 provides a logical view of the Network/communication Layer of nSHIELD focusing on the services that should be supported.

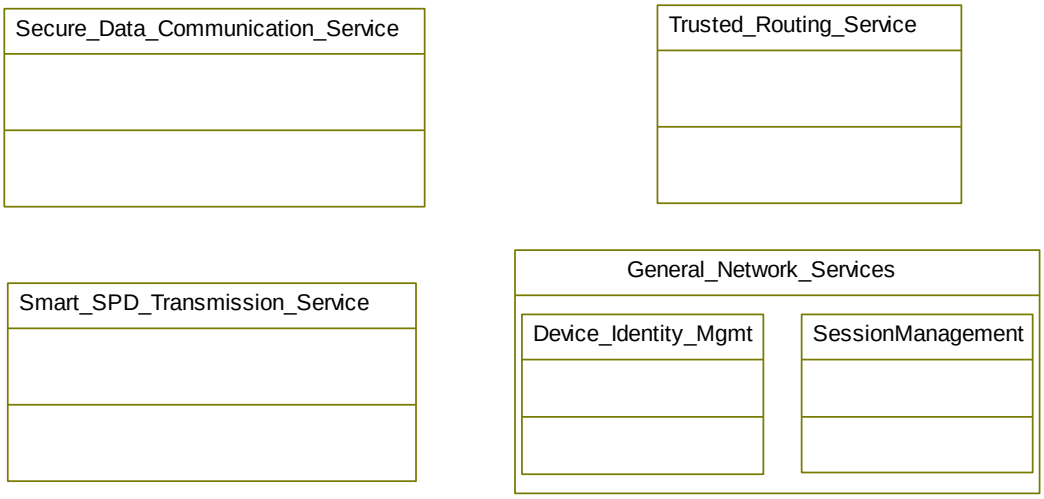

Figure 6-7: nSHIELD's Network/Communication Layer Services

6.3.1.1 Trusted and dependable connectivity

Trusted and dependable connectivity should be regarded at two different levels:

- A *Trusted Network Routing Service*
- A *Secure Data Exchange/Communication Service*

Trusted Network Routing Service

In terms of network routing, an nSHIELD node should implement one or more trusted routing schemas (protocols). Considering a distributed approach, these protocols are based on trust models that consider a number of measurements (metrics) for routing purposes. Direct or indirect measurements[3] can be taken into account for evaluating trust value for a node. Reputation-based schemas, relying on the combination of the trust value calculated locally with the trust values calculated by other nodes, are considered to provide higher protection than simple direct measurement based trust evaluation. Such schemas are mainly used in wireless networking to provide secure routing functionality. In distributed systems, where there is no central infrastructure to implement full communication among all participants, each individual entity must depend on its neighbors to carry out its transactions. Due to the open medium and the dynamic entrance of new nodes to such networks, there must be a way to establish trust relationships to avoid malicious users. Reputation is formed by a node's past behavior and reveals its cooperativeness. A node with a high reputation level can be considered as trustworthy. Legitimate nodes depend mostly on trustworthy entities to accomplish communication tasks, like routing and forwarding. Furthermore, low reputation can reveal selfish or malicious entities and is used for intrusion detection. Legitimate nodes try to avoid such entities and do not forward their traffic. There are three main goals that a reputation-based scheme is trying to accomplish:

[3] Direct measurements are measurements performed by the node itself. All nodes in the network monitor the behavior of their adjacent nodes and compute a direct trust level for them based upon their sincerity in execution of the routing protocol. On the other hand, indirect measurements are the corresponding measurements performed by the other nodes. A combination of direct and indirect observations (reputation-based schema) provides higher protection than simple direct measurement based trust evaluation.

- To provide information to distinguish between a trustworthy entity and an untrustworthy one
- To encourage entities to act in a trustworthy manner
- To discourage untrustworthy entities from participating in the system

A trusted routing service requires communication of control information between the nodes and thus the implementation of a specific protocol (type of messages, exchanged frequency, interactions, etc.). Moreover, it may affect the performance of the node, and network in general, since it will consume resources for transmission and processing. Networks with ultra-constrained devices may not be able to support heavy reputation-based schemas that offer high levels of security

In general, the trusted routing schema is not supposed to completely replace the existing network routing protocol of the node. Rather it runs on top of legacy routing algorithms (i.e. IP), having the capability to modify the routing tables info based on the confidence level evaluated for its neighbours.

The set of metrics considered for establishment of trust might be configurable based on requests/commands from higher layers (i.e. middleware). Generally these metrics should be a subset of the defined network layer SPD metrics. However nodes energy constraints or even policy constraints may also be taken into account.

All nSHIELD devices should implement at least one trusted routing schema/protocol in common. Considering that multiple implementations of reputation-based protocols are available on a device we can think of an additional feature that enables the dynamic selection of a trusted protocol on request, or based on a set of SPD metrics. This will require the existence of an appropriate software control module that could handle all the necessary actions.

Secure Data Exchange/Communication Service

The network layer should provide mechanisms that allow for secure network access and safe exchange of data over the nSHIELD network. This may include:

- Encryption schemes to enable protection (and integrity) of data
- CRC encoding and checksum techniques to verify data integrity
- Authentication schemes to verify identity of sender/receiver

In order to confront security risks and ensure privacy and data integrity in all parts of the network, nSHIELD nodes shall implement *Encryption* schemes. The implementation of the specific nSHIELD encryption/decryption functionalities will be based on the evaluation and classification of cryptosystem attacks and the corresponding selection of cryptography type (e.g. Public Key Vs Symmetric). The cryptography framework is determined and handled by a key management scheme, charged with the generation, distribution, storage and use of keys. The key-certification mechanism is the most critical procedure in cryptosystems and highly interdependent to system architecture. The capability of nSHIELD devices to implement cryptography technologies comes with respective trade-offs, especially in reference with node resources, such as computational load, memory and energy. For the encryption of data in nSHIELD we foresee a dedicated component called *Crypto Manager*.

Authentication is the complementary to encryption security component, vital especially in the context of wireless transmission. Compared to encryption, authentication seems to be more multifaceted in terms of selecting an appropriate scheme. It has to ensure that only authorized users exchange information, focusing in the verification of the sender's (or receiver's) identity. The network is protected against unauthorized access and use, usually through the use of credentials, such as passwords, keys or digital certificates. Again, the suggested nSHIELD authentication protocols will be based on the types of network entities and communications and the security requirements imposed. Authentication models perform their task through a sequence of exchanged messages between the involved parties (e.g. authenticated supplicant and authenticator). These messages contain keys, which are mathematically modified through an iterative process that leads the sender and receiver sharing eventually a common session key. For the processing of authentication mechanisms in nSHIELD we describe the establishment of dedicated *Sessions,* which are kept "alive" for a given time interval, so as to minimise the overhead that would

otherwise be required for repeated session negotiation procedures. Data integrity mechanisms incorporated in the cryptographic protocol in use will ensure that no data modifications have taken place (both intentional and unintentional) during the session lifetime.

6.3.1.2 Smart SPD transmission

An nSHIELD node should support a service that allows smart (and secure) transmission of data based on SPD built-in features at node level. This is considered a communication layer feature that relates not only with the traditional network layer of the OSI model but also with the physical layer. The implementation of the smart SPD transmission service should be based on the basic principles of Software Design Radio (SDR) and Cognitive Radio (CR) systems.

Through the *smart SPD transmission service* an nSHIELD device will be able to provide reliable and efficient communications even in critical (physical) channel conditions by using adaptive and flexible algorithms for dynamically configuring and adapting various transmission related parameters (i.e. type of modulation, type of coding, use of multiple antennas, used frequency, transmission power levels, bandwidth rate, etc.).

6.3.1.3 Other Network Services

At network layer, we should consider also a couple of additional services that although do not directly relate to SPD capabilities they are considered necessary in order to support communication within the nSHIELD network. These services are:

- A *Session Management Service*
- A *Device Identity Management Service*

Session Management Service

This service is responsible to keep track and manage sessions within the network communications inside the nSHIELD system. A session is set up or established at a certain point in time, and torn down at a later point in time. Sessions are possibly needed to support and synchronize stateful communication between nSHIELD devices where more than one messages are needed in each direction.

Device Identity Management Service

The role of this service is to provide a unified addressing schema for all embedded devices participating in an nSHIELD network. The implemented mechanism should permit the assignment and management of devices' IDs independent of their physical addresses thus hiding the heterogeneity of the nodes. These assigned IDs can be utilized by other services within or outside the network layer. For example the service discovery service at middleware layer can return a set of such IDs uniquely identifying a device. Network services like the trusted routing service can exploit the unified addressing for enabling trusted routing among devices that use different addressing schemas.

6.3.2 Development View

The network/communication layer of nSHIELD cannot be regarded as an entity consisting purely of software modules and therefore a development view diagram cannot properly capture its aspects. The development view however, is used only to visualize a more fine grained view of the various entities involved for the realization of the services. These entities are not limited to software modules but may also include complete protocols or simple functions that perform a dedicated task. This is depicted in Figure 6-8 where important dependencies between entities internal or external to the Network/communication layer are also drawn. Based also on the description of services, performed in section 6.3.1, a list of the elements that are required for the implementation of services that directly related to SPD capabilities is provided. For each such service we briefly state possible external or internal dependencies.

Secure Data Exchange/Communication Service

Encapsulates/implements appropriate functions that perform:

- *Encryption* of data to be send (data security)
- *Decryption* of data received (data security)
- *Error-checking techniques* i.e. (*CRC encoding)* of data send using protocols that do not inherently support that for their messages (data integrity)
- *Authentication*\verification of data source or destination (communication security)

Dependencies:

The implementation of these functions may require support of external to the network layer modules. For example in most of the above cases there is a requirement for an entity that can provide cryptographic keys as well as for a mechanism to securely distribute them within a network of ESDs.

Another identified dependency is with the Session Management Service in case authentication of both sender and recipient is needed.

Trusted Network Routing Service

Encapsulates/implements routing protocols that implement Reputation-based or Direct Trust algorithms.

Dependencies:

The trusted routing algorithm may relay on the value of metrics that are provided by another functional layers (most notably from node layer)

Routing tables of existing routing protocols (i.e. IP) may be reused and modified by the trusted routing service.

Device Identity Management Service may be contacted to provide unique nSHIELD IDs needed by the algorithms.

Smart SPD transmission

The service implies the existence of two important sub-elements:

- A *sensing module* that is able to gather information that will permit to achieve awareness of the radio spectrum and physical layer capabilities as well as of a node's resources. Information can be provided on a periodic or on demand basis and usually implies an interface with appropriate node layer services. The exact type and number of parameters to be considered for monitoring will depend on the sensing capabilities of the underlying hardware but also on requests from higher layers.
- A *reconfiguration/parameter adjustment module* that can utilize the information gathered above and take decisions (reason) on reconfigurations or adjustments that need to apply regarding physical transmissions attributes and nSHIELD network layer configuration in general. The reasoning can be based on a set of provided (SPD) metrics according to the required QoS.

Dependencies:

The two modules may have inter-dependencies.

External dependencies are not foreseen although both modules may need to interface with node layer components.

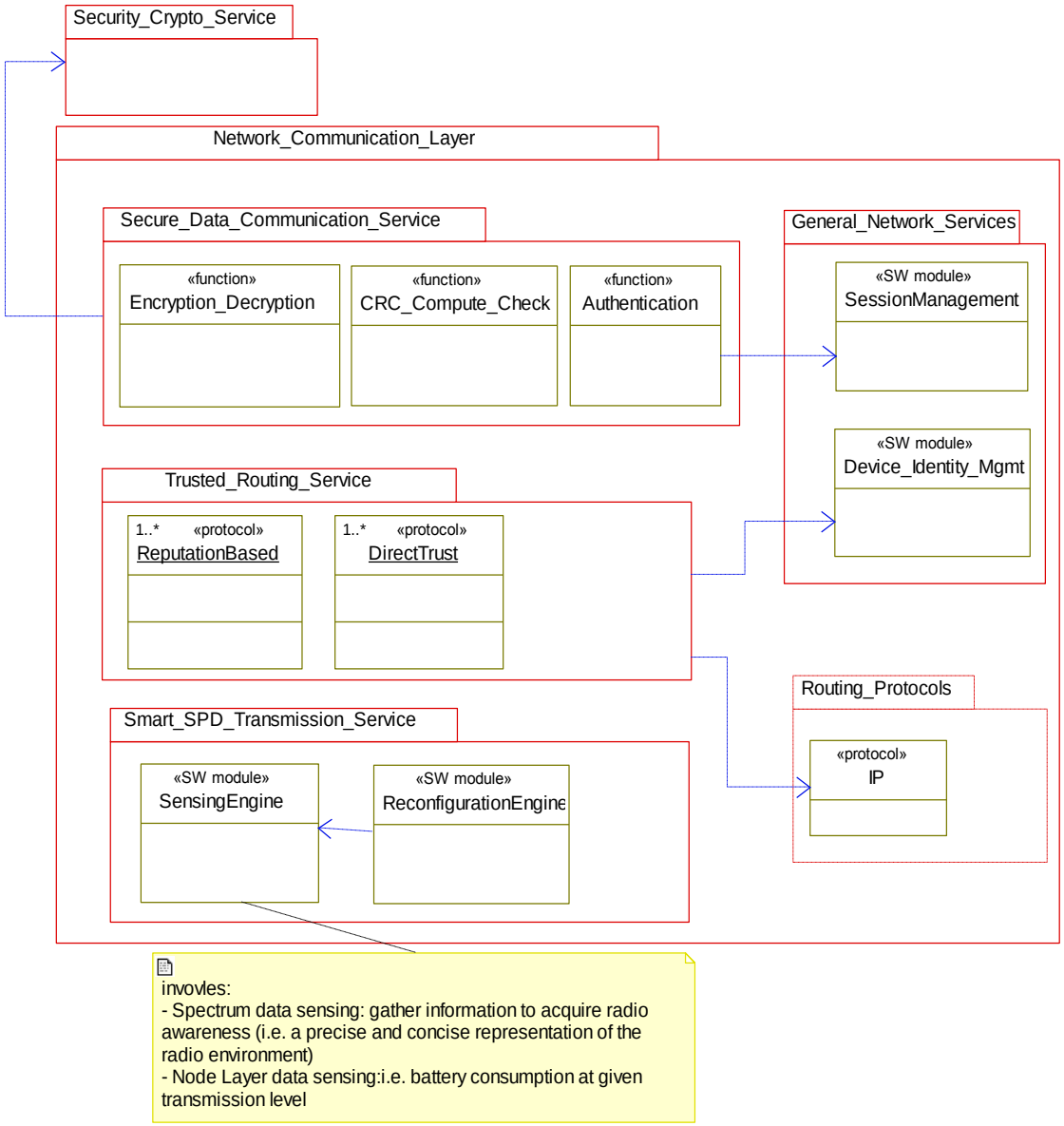

Figure 6-8: nSHIELD's Network/Communication Layer Services (Logical View)

6.4 Middleware

The SHIELD Middleware is basically a software layer installed in the SHIELD nodes in different versions depending on the available HW capabilities (complex middleware for high capacity nodes, lightweight middleware for less-performing nodes). This software act as a glue for the different SPD services offered by the SHIELD system, (node, network and middleware layer itself) since it allows to: abstract, discover, compose and control them, by means of dedicated protocols, control algorithms and interfaces.

The middleware component is a *mandatory* component to be supported by all types of nSHIELD nodes (nS-ESD, ns-ESD GW, nS-SPD-ESD). In the following section the logical view of the middleware will be provided, as well as some preliminary considerations on the development and deployment view.

6.4.1 Logical View and Services Description

On a logical/functional point of view, the SHIELD Middleware is based on two categories of *services:*

- *Middleware Core SPD services*, i.e. services available in a generic middleware and necessary to its correct behaviour, including services necessary to realise the dynamic composability of SPD functionalities (otherwise the middleware will not be a 'SHIELD' middleware)
- *Innovative SPD Services,* i.e. services that provide (enriched) SPD functionalities and enable interoperability with legacy devices

These sets may even overlap, in case a core service is enriched with SPD features (for example discovery is a core service, but it can become an SPD service by implementing a 'secure discovery')

The list of the main middleware services is summarized in the following figure. This list may not be exhaustive since, during the prosecution of the project, more functionality could be added, depending on needs and possibilities. However the modular (plug&play) architecture of the middleware allows a seamless introduction of new services.

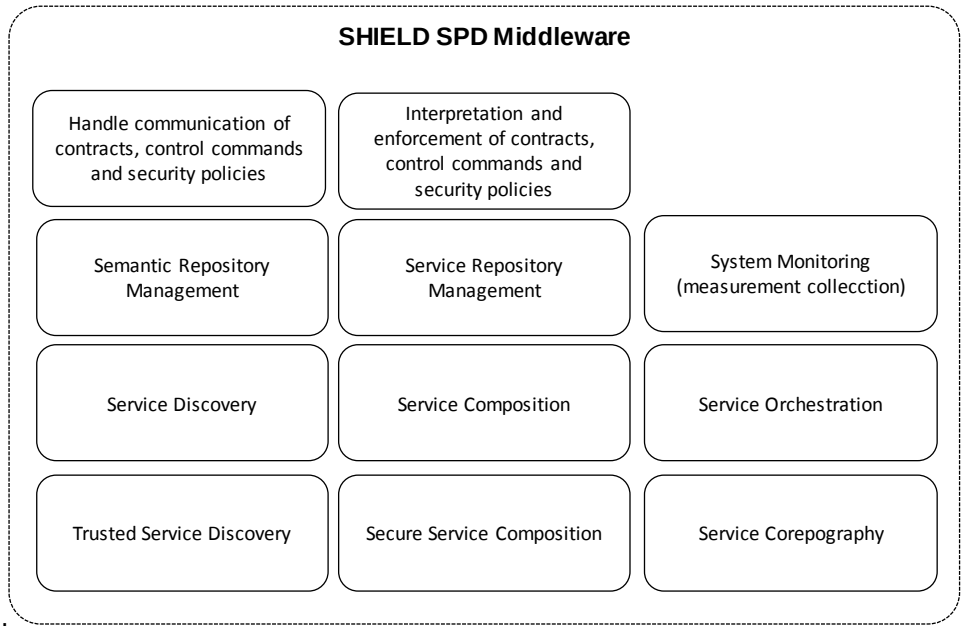

Figure 6-9: SHIELD Middleware services and functionalities

6.4.1.1 (Secure) Service discovery

This service allows any SHIELD Middleware Adapter to discover the available SPD functionalities and services over heterogeneous environment, networks and technologies that are achievable by the nSHIELD Embedded System Device (nS-ESD) where it is running. Indeed the pSHIELD secure service discovery uses a variety of discovery protocols (such as SLP[4], SSDP[5], NDP[6], DNS[7], SDP[8], UDDI[9]) to

[4] IETF Service Location Protocol V2 - http://www.ietf.org/rfc/rfc2608.txt

[5] UPnP Simple Service Discovery Protocol - http://upnp.org/sdcps-and-certification/standards/

[6] IETF Neighbour Discovery Protocol - http://tools.ietf.org/html/rfc4861

[7] IETF Domain Name Specification - http://www.ietf.org/rfc/rfc1035.txt

[8] Bluetooth Service Discovery Protocol

harvest over the interconnected Embedded System Devices (ESDs) all the available SPD services, functionalities, resources and information that can be composed to improve the SPD level of the whole system. In order to properly work, a discovery process must tackle also a secure and dependable service registration, service description and service filtering. The service registration consists in advertising in a secure and trusted manner the available SPD services. The advertisement of each service is represented by its formal description and it is known in literature as service description. The registered services are discovered whenever their description matches with the query associated to the discovery process, the matching process is also known in literature as service filtering. On the light of the above a SPD services discovery framework is needed as a core SPD functionality of a nSHIELD Middleware Adapter. Once the available SPD services have been discovered, they must be prepared to be executed, assuring that the dependencies and all the services preconditions are validated. In order to manage this phase, a service composition process is needed.

6.4.1.2 (Trusted) Service composition

This service is in charge to select those atomic SPD services that, once composed, provide a complex and integrated SPD functionality that is essential to guarantee the required SPD level. The service composition is an nSHIELD Middleware Adapter functionality that cooperates with the nSHIELD Overlay in order to apply the configuration strategy decided by the Control Algorithms residing in the pSHIELD Security Agent. While the Overlay works on a technology independent fashion composing the best configuration of aggregated SPD functionalities, the service composition takes into account more technology dependent SPD functionalities at Node, Network and Middleware layers. If the Overlay decides that a specific SPD configuration of the SPD services must executed, on the basis of the services' description, capabilities and requirements, the service composition process ensures that all the dependencies, configuration and pre-conditions associated to that service are validated in order to make all the atomic SPD services to work properly once composed.

Composition may be enriched by making it trusted. The proper service selection is difficult when there are many candidate services in a service repository. Usually the minimum requirements, like functional attributes, are satisfied by many services. Thus other non-functional features like trust should be introduced in the selection process. Trust refers to several factors such as quality, reputation, cost, availability and experience. The trust factors must be specified in service description to ease the service discovery phase. Trust is a complex factor and it can take many forms such as belief, honesty, truthfulness, competence, reliability and confidence or faith of the service provider, consumer, agents and service. Specific algorithms and procedures take cares of this aspect.

6.4.1.3 Service orchestration

This functionality is in charge to deploy, execute and continuously monitor those SPD services which have been discovered and composed. This is part of the SHIELD Middleware Adapter functionality. While service composition works "off-line" triggered by an event or by the SHIELD Overlay, service orchestration works "on-line" and is continuously operating in background to monitor the SPD status of the running services.

This functionality performs basically the same tasks as the orchestration, with the main difference that in this approach there is no a single engine that takes the control of the entry point (centralized approach), and so that interactions are peer-to-peer. Control is not established by neither of the entry nor sequential points that are being processed. This enables the capacity to dynamically reorganise and be tolerant to unexpected happenings.

6.4.1.4 Semantic Repository Management

This service is responsible of managing any semantic information related to the SHIELD components (interface, contract, SPD status, context, etc.). The use of common SPD metrics and of a shared ontology to describe the different SPD aspects involved in guaranteeing a precise level of SPD, allows to dominate

[9] OASIS Universal Description Discovery and Integration - http://www.uddi.org/pubs/uddi_v3.htm

the intrinsic heterogeneity of the SPD components. Any semantic data is thus technology neutral and it is used to interface with the technology independent mechanisms applied by the SHIELD Overlay.

6.4.1.5 Service Repository Management

This functionality act as a database to store the service entries (e.g. the SPD components description of provided functionalities, interfaces, semantic references, etc.) used by the choreographer/orchestrator and by the enforcement engines to actuate decisions. Any SHIELD Node, Network or Middleware layer component can be registered here to be discovered.

6.4.1.6 Handle communication of contracts, control commands and security policies

This functionality is the direct link with the SHIELD Overlay and is responsible of sending ot it all the information necessary to take “intelligent” decision on composition of SPD functionalities to reach the SPD objectives. In some cases information of SPD components are sent to control algorithms, and in some others contract or policies are forwarded for more structured decision.

6.4.1.7 Interpretation and enforcement of contracts, control commands and security policies

The decisions taken by the SHIELD Overlay in terms of control commands, policies, enforcement actions and so on are received by the middleware and must be translated into ‘actions’ on the underlying systems (node, network and middleware layer itself) in terms of, for example, protocols activation, parameters configurations and so on. This is in charge of the interpretation and enforcement functionality.

6.4.1.8 System monitoring (measurement collection)

Together with services information and semantic information, also measurements could be provided to the Overlay to take composition decision. This task is in charge to a dedicated functionality (defined during the prosecution of the project) that can monitor both internal and/or external parameters. This is complementary to discovery functionality and, since it requires more resources, its feasibility on SHIELD different hardware.

6.5 Overlay

The Overlay is a logical vertical layer in charge of deciding, according to control algorithms or policies, which SPD functionalities should be activated/ deactivated and to tailor them in order to reach the SHIELD objectives (i.e. a desired SPD level). This layer is indeed a software routine running over the SHIELD Middleware and using the Middleware core services to collect information and actuate its decision. Its logical/functional view is described in the following.

6.5.1 Logical View and Services Description

The SHIELD Overlay offers five services, partially overlapped with the middleware ones. They are depicted in the following figure.

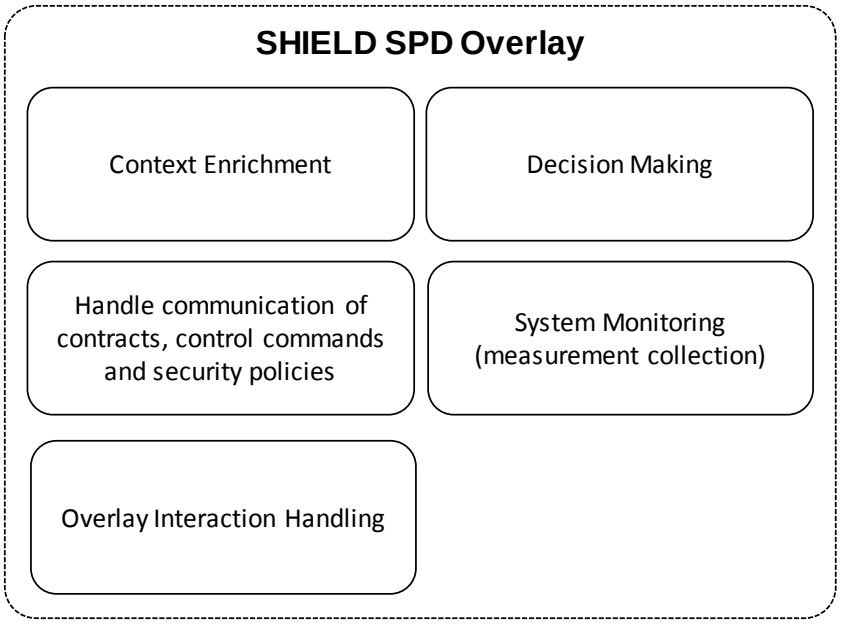

Figure 6-10: SHIELD Middleware services and functionalities

In particular System Monitoring and Handling of communication of contracts, control commands and security policies are also present in the middleware. This is not impossible, because all of them are software routines and the boundaries are not fixed. For the moment they are duplicated in both the elements and in the prosecution of the project the architectural analysis will decide on which layer they should be included.

6.5.1.1 System monitoring

This service is in charge to interface the Overlay layer with the Middleware layer, to retrieve sensed metadata from heterogeneous nSHIELD devices belonging to the same subsystem, to aggregate and filter the provided metadata and to provide the subsystem situation status to the context engine.

6.5.1.2 Context enrichment

This service is in charge to keep updated the situation status as well as to store and maintain updated any additional information exchanged with other SPD security agents that are meaningful to keep track of the situation context of the controlled nSHIELD subsystem. The situation context contains both status information and configuration information (e.g. rules, policies, constraints, etc.) that are used by the decision maker engine.

6.5.1.3 Decision making

This functionality uses the valuable, rich input provided by the context engine to apply a set of adaptive (closed-loop, rule-based or policy based) and technology-independent algorithms. The latter, by using (as input) the above-mentioned situation context and by adopting appropriate advanced methodologies able to profitably exploit such input, produce (as output) **decisions** aiming at guaranteeing, whenever it is possible, target SPD levels over the controlled nSHIELD subsystem.

6.5.1.4 Handle communication of contracts, control commands and security policies

The decisions mentioned above are translated into a set of proper enforcement rules actuated by the nSHIELD Middleware layer all over the nSHIELD subsystem controlled by the considered SPD Security Agent. These rules are sent to the Middleware via this functionality

6.5.1.5 Overlay Interaction Handling

Since the SHIELD System is supposed to be composed by hundreds of nodes and SPD functionalities, in order to improve overall performances it has been planned to cluster the system into segments (possibly standalone), each one managed by a single Security Agent (i.e. the software entity that performs Overlay tasks). Even if clusters are stand alone, interactions among them shall be foreseen to coordinate actions or to enrich the knowledge basis to take decision. This is in charge to the overlay interaction handling.

6.5.2 Development and Deployment view

In compliance with the overall nSHIELD Architecture depicted in Figure 6-4 and based on the internal structure of the nS-SPD-ESD node as depicted in Figure 6-5, the nSHIELD overlay functionality is implemented through a *security agent* component. This component actually controls a given nSHIELD Subsystem. Expandability of such framework is obtained by enabling communication between SPD *Security Agents* controlling different sub-systems through the provided overlay interface. Therefore, the presence of more than one SPD Security Agents is justified by the need of solving scalability issues in the scope of system-of-systems (exponential growth of complexity can be overcome only by adopting a hierarchical policy of divide et impera). Within an nSHIELD subsystem multiple security agents could be possible mainly for redundancy or high availability purposes (usually only one will be active).

Each SPD Security Agent, in order to perform its work, exchanges carefully selected information with the other SPD Security Agents, as well as with the three horizontal layers (node, network and middleware) of the controlled nSHIELD subsystem. Each SPD Security Agent collects properly selected heterogeneous SPD-relevant measurements and parameters coming from node, network and middleware layers of the controlled nSHIELD subsystem. The SPD Security Agent is a software module and requires the mediation of the nSHIELD Middleware. Thus any actual communication between the Overlay and the three layers is performed passing physically through the middleware layer.

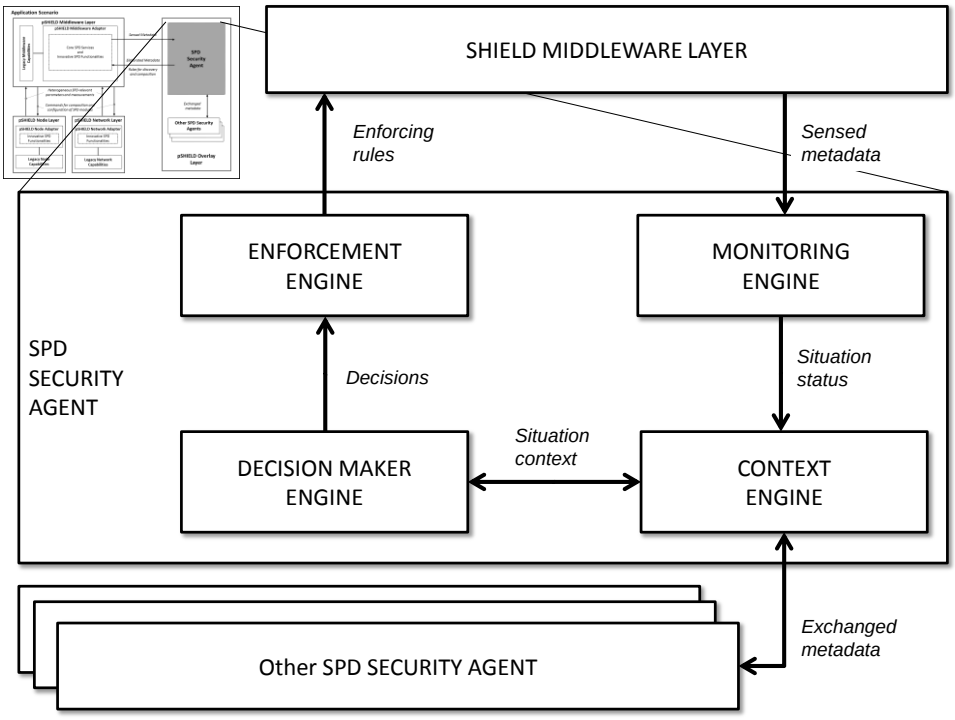

Figure 6-11: nSHIELD SPD Security agent architecture

The heterogeneous data collected from the three horizontal layers (passing through the middleware layer) are abstracted and translated into technology-independent metadata. The resulting metadata (referred to as **sensed metadata**) are interpreted by the monitoring engine and stored in the context engine.

The **monitoring engine** is in charge to interface the Overlay layer with the Middleware layer, to retrieve sensed metadata from heterogeneous nSHIELD devices belonging to the same subsystem, to aggregate and filter the provided metadata and to provide the subsystem situation status to the context engine.

The **context engine** is in charge to keep updated the situation status as well as to store and maintain updated any additional information exchanged with other SPD security agents that are meaningful to keep track of the situation context of the controlled nSHIELD subsystem. The situation context contains both status information and configuration information (e.g. rules, policies, constraints, etc.) that are used by the decision maker engine.

The **decision maker** engine uses the valuable, rich input provided by the context engine to apply a set of adaptive (closed-loop or rule-based) and technology-independent algorithms. The latter, by using (as input) the above-mentioned situation context and by adopting appropriate advanced methodologies able to profitably exploit such input, produce (as output) **decisions** aiming at guaranteeing, whenever it is possible, target SPD levels over the controlled nSHIELD subsystem.

The decisions mentioned above are translated by the **enforcement engine** into a set of proper **enforcement rules** actuated by the nSHIELD Middleware layer all over the nSHIELD subsystem controlled by the considered SPD Security Agent.

7 Interfaces

This section attempts to register and categorize the interfaces which occur in an nSHIELD system. The term is quite generic; In general as interfaces we should not consider physical interconnections (i.e. Ethernet, etc.). Rather the focus should be given in logical interconnectivity that includes mainly the data and information that flows or is exchanged within the nSHIELD system. and possibly to the APIs that are provided or required by the various nSHIELD software modules that exist at the 4 nSHIELD functional layers. Based on this approach interfaces should be considered on various levels depending on the type of elements that are involved:

- ***Internal node interfaces:*** that address data and interactions that occur between the 4 functional layers within a single ESD
- ***External node interfaces:*** that address data and interactions that may occur between the 4 functional layers of different ESDs
- ***Components' or intra-layer interfaces:*** that include the interfaces that may exist between the various components implementing an nSHIELD functional layer

It must be noted that interfaces description addresses information and data exchange that is needed to address nSHIELD capabilities and support the foreseen SPD services. Therefore we do not consider application specific flows (i.e. sensor data).

7.1 Internal

This section describes the information that flows between layers within a device. For each functional layer brief information is provided regarding incoming and outgoing data flows in the following table. The **(→)** symbol in each layer of Table 6-1 denotes that the row cells provide outgoing information flows (towards the functional layers depicted as column titles).

Table 7-1: information flows between the various nSHIELD layers (within a device)

Provided interfaces	Node layer	Network layer	Middleware layer	Overlay layer
Node layer (→)	–	**Measurements** (on request or periodically): • Metric values to be used by *trusted routing service* reputation based scheme (i.e. battery lifetime, RSSI) • Radio environment data to be used by smart transmission service (i.e. available resources, number of active users etc.) **Secure Keys** from crypto key generator module to be used from secure data communication service	**Measurements** (On request or periodically to compute SPD metrics) **TBD in more details**	–
Network layer (→)	**Commands** • To configure/ reconfigure transmission related parameters of the RF channel (smart transmission service) • …	–	**SPD metrics** (available at network level) nSHIELD Device Unique Network Id **TBD in more details**	–

Middleware layer (→)	**Commands** (for composition or configuration of SPD modules) **TBD in more details**	**Commands** • To control and configure (i.e. define the metrics used) the trusted routing schema used • To configure parameters related to data encryption (i.e. type of algorithm to be used • ... **Other TBD**	–	**sensed (SPD) metadata** **TBD in more details**
Overlay layer (→)	–	–	**Rules for compostion and discovery** **TBD in more details**	–

Table 7-1 consists a first approach in defining the type of information exchanged between the various nSHIELD functional layers within an embedded node. It is expected to be refined and detailed in future versions of the architecture deliverable.

7.2 External

This section describes the information that flows between layers of different devices. For each functional layer brief information is provided regarding incoming and outgoing data flows in the following table. The **(→)** symbol in each layer of Table 7-2 denotes that the row cells provide outgoing information flows (towards the functional layers depicted as column titles).

Table 7-2: information flows between the various nSHIELD layers (between different ESDs)

Provided interfaces	Node layer	Network layer	Middleware layer	Overlay layer
Node layer (→)	–	–	–	–
Network layer (→)	–	Control packets needed from applied trusting routing protocol **Other TBD**	**??**	–
Middleware layer (→)	–	–	**TBD**	**TBD**
Overlay layer (→)	–	–	**TBD**	**TBD**

Table 7-2 consists a first approach in defining the type of information exchanged between the various nSHIELD functional layers between different embedded nodes. The table currently does not provide much information. Most of its cells are empty and we include it mainly with the purpose to provide a first suggestion regarding between which functional layers information flows exist. It should be reviewed, refined and detailed in future versions of the architecture deliverable.

7.3 Components

Information regarding data flows and interfaces between components of a specific nSHIELD functional layer is not provided in this version of the architecture document. The present document provides only a preliminary version of the architecture, presenting in most cases only a logical functional view together with the description-analysis of the capabilities and services available in each layer. Therefore, no adequate information existed in order to proceed with a detailed specification of the interfaces between

the various elements implementing a specific layer. Information on components' interfaces will be provided in future versions of the architecture deliverable while other views of the nSHIELD's functional layers will become available.

8 Application Scenarios Realization

Since this document provides only a preliminary definition of the nSHIELD architecture, it is in not possible to proceed, at this stage, with specific realizations of the system architecture based on the envisaged application scenarios. Currently, the 4 nSHIELD scenarios (Railroad Security, Voice/Facial Verification, Dependable Avionic System, Social Mobility and Networking) were used mainly to identify major needs, in terms of expected functionalities and SPD capabilities, and through them drive the definition of a generic architecture that could provide full support for them. The goal is the final reference architecture to be generic enough in order to support all possible application scenarios. If this is not possible, then future versions of the architecture deliverable will provide here detailed use cases that will facilitate the realization of the nSHIELD architecture for an application scenario. Such use cases may contain the following data:

- Description
- Involved persons (Users/Clients, Authorities, Personnel)
- Services
- Policies
- Problems
- Management

9 Conclusions

D2.3 reflects the efforts to yield a first preliminary nSHIELD architectural framework. This included setting the background knowledge, defining basic terms and work methodology, identifying substantial requirements and generally trying to take into account, to a big or lesser extent, all input available at this stage of the project.

The outcome is the outline of an overall architecture scheme, starting from the definition of component devices. The proposed architecture introduces 3 new types of embedded devices: *nS-ESD*, *nS-ESD GW* and *nS-SPD-ESD* to address the definition of an nSHIELD system and its interaction with the rest of the world (through what we call *legacy* devices). Inversely, the hierarchical logical view of the conceptual architecture demands the introduction of two more group concepts: the *nSHIELD Subsystem* containing one or more *nSHIELD Aware Clusters*. To implement this scheme, in terms of physical nodes, the previously introduced three categories are preserved: *Nano*, *Micro/Personal* and *Power Nodes*.

According to nSHIELD documents and work plan, the general architecture is decomposed to four functional layers, for each of which the logical view and a basic set of services are described. The *Node* layer provides SPD functionalities. The *Network* layer is charged with trusted connectivity and smart SPD transmission. The *Middleware* layer is responsible for all the stages of services management. Finally the *Overlay* layer based on the concept of *Security Agent*, controls subsystems, manages their communication and organizes the composability of different SPD technologies and modules.

Conclusively, this document depicts formalized (as possible) system architecture and sets the technical challenges, the open issues and subsequent work plan. Continuing in T2.3 the proposed structure will be refined. Interfaces, mentioned here in a preliminary level, shall be clearly defined to validate the internal robustness and external communication of the system. As nSHIELD project aims at creating an impact on the SPD market of ESs, it is essential to link the theoretical architectural framework with real implementations, providing thus its proof of concept. In this context, nSHIELD Architecture will be realized through instances of the four application domains registered in the DoW and close interaction with the activities of WP6 (Platform integration, validation & demonstration) is deemed necessary

10 References

[1] The pSHIELD project web site http://www.pshield.eu/
[2] pSHIELD-D2.3.2_System_architecture_design, pSHIELD project official deliverable, Jan. 2012
[3] nSHIELD_D2.2_ Preliminary System Requirements and Specifications, nSHIELD project deliverable, May 2012
[4] OSGi Alliance. Osgi service platform specification overview. http://www.osgi.org/resources/spec overview.asp, 2003.
[5] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid. http://www.gridforum.org/ogsi-wg/drafts/ogsa draft2.9 2002-06-22.pdf, 2003.
[6] Sun Microsystems. Jini network technology. http://wwws.sun.com/software/jini/, 2003.
[7] W3C. Web services architecture. http://www.w3.org/TR/2003/WD-ws-arch-20030808/, 2003
[8] Bertrand Meyer. Contracts for components. Software Development, 2000.
[9] Antoine Beugnard. Jean-Marc jezequel et al. Making Components Contract Aware, IEEE Journal Computer archive Volume 32 Issue 7, July 1999
[10] Tammy Noergaard, Embedded Systems Architecture, A Comprehensive Guide for Engineers and Programmers, Elsevier Inc. 2005
[11] http://www.middleware.org/whatis.html
[12] The official CORBA standard from the OMG group – http://www.omg.org/spec/CORBA/Current/
[13] DCOM Remote Protocol Specification – http://msdn.microsoft.com/library/cc201989.aspx
[14] Nikola Milanovic, Jan Richling, Miroslaw Malek, Lightweight Services for Embedded Systems, Proceedings of the 2nd IEEE Workshop on Software Technologies for Embedded and Ubiquitous Computing Systems (WSTFEUS 2004), Vienna, Austria, 2004
[15] The Hydra project – http://www.hydramiddleware.eu
[16] Prism-MW - Architectural Middleware for Mobile and Embedded System – http://csse.usc.edu/~softarch/Prism/
[17] http://www.embeddedlibrary.com/Embedded_Science/Embedded_Systems/Embedded_Systems_Design_and_Development_Lifecycle.html
[18] IEEE 1471 "Recommended Practice for Architectural Description of Software-Intensive Systems"
[19] IEEE ISO/IEC 42010:2007, Systems and Software Engineering---Recommended practice for architectural description of software-intensive systems
[20] Ministry of Defense Architecture Framework (MoDAF), http://www.mod.uk/DefenceInternet/AboutDefence/CorporatePublications/InformationManagement/MODAF/ModafMetaModel.htm
[21] Department of Defense Architecture Framework (DoDAF), http://dodcio.defense.gov/dodaf20.aspx
[22] The Open Group Architecture Framework (TOGAF), http://www.togaf.info/
[23] ISO Reference Model for Open Distributed Processing (RM-ODP), http://www.rm-odp.net/
[24] 4+1 View Model, http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf
[25] A. J. Goldsmith and S. B. Wicker, “Design Challenges for Energy-Constrained Ad Hoc Wireless Networks,” *IEEE Wireless Communications Magazine*, pp. 8–27, Aug. 2002.
[26] S. Shakkottai, T. S. Rappaport, and P. C. Karlsson, “Cross-Layer Design for Wireless Networks,” *IEEE Communications Magazine*, vol. 41, no. 10, pp. 74–80, Oct. 2003
[27] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. New Jersey: Prentice Hall, 1992.
[28] V. T. Raisinghani and S. Iyer, “Cross-Layer Design Optimizations in Wireless Protocol Stacks,” *Computer Communications*, vol. 27, pp. 720–724, 2004.
[29] A. S. Tanenbaum, *Computer Networks*, 3rd ed. Prentice-Hall, Inc., 1996.
[30] L. Larzon, U. Bodin, and O. Schelen, “Hints and Notifications,” in *Proc. IEEE Wireless Communications and Networking Conference (WCNC’02)*, Orlando, 2002.
[31] G. Xylomenos and G. C. Polyzos, “Quality of service support over multiservice wireless internet links,” *Computer Networks*, vol. 37, no. 5, pp. 601–615, 2001.
[32] Q. Wang and M. A. Abu-Rgheff, “Cross-Layer Signalling for Next-Generation Wireless Systems,” in *Proc. IEEE Wireless Communications and Networking Conference (WCNC’03)*, New Orleans, 2003.
[33] M. Conti, G. Maselli, G. Turi, and S. Giordano, “Cross-Layering in Mobile Ad Hoc Network Design,” *IEEE Computer Magazine*, pp. 48–51, Feb. 2004.

[34] Soon-Hyeok Choi, Dewayne E. Perry and Scott M. Nettles: “A Software Architecture for Cross-LayerWireless Network Adaptations”, The University of Texas at Austin Austin, Texas
[35] Vijay T. Raisinghani, Sridhar Iyer: “Cross Layer Feedback Architecture for Mobile Device Protocol Stacks”,
[36] Vineet Srivastava and Mehul Motani Srivastava: “Cross-Layer Design: A Survey and the Road Ahead”, IEEE Communications Magazine, Dec 2005.
[37] Giovanni Giambene and Sastri Kota: “Cross-layer protocol optimization for satellite communications networks: A survey”, Int. J. Satell. Commun. Network. 2006
[38] Qi Wang and Mosa Mi Abu-Rgheff: “A Multi-Layer Mobility Management Architecture Using Cross-Layer Signalling Interactions” http://www.cis.udel.edu/~yackoski/cross/qwang_epmcc03_paper.pdf
[39] R. Winter et al., “CrossTalk: A Data Dissemination-Based Crosslayer Architecture or Mobile Ad Hoc Networks,”
[40] V. T. Raisinghani and S. Iyer: “ECLAIR: An efficient cross layer architecture for wireless protocol stacks”, WWC2004,
[41] Yana Bi, Mei Song, Junde Song: “Seamless mobility Using Mobile IPv6”, Publication Year: 2005
[42] Shantidev Mohanty and Ian F. Akyildiz: “A Cross-Layer (Layer 2 + 3) Handoff Management Protocol for Next-Generation Wireless Systems” IEEE Transactions on Mobile Computing, vol. 5, no. 10, Oct 2006
[43] Melhus, I.,Gayraud, T., Nivor, F., Gineste, M., Arnal, F., Pietrabissa, A., Linghang Fan: “SATSIX Cross-layer Architecture” Publication Year: 2008 , Page(s): 203 – 207
[44] Srivastava, V., Motani, M.: “The Road Ahead for Cross-Layer Design”, Publication Year: 2005 , Page(s): 551 – 560.
[45] IETF: Policy Framework [Online], http://datatracker.ietf.org/wg/policy/charter/
[46] Verma D.C.: “Simplifying network administration using policy-based management”, IEEE Network, 2002, pp. 20-26
[47] ETSI DES 282 001 V0.0.1 (2006-09)
[48] M. Al-Kuwaiti et al., “A Comparative Analysis of Network Dependability, Fault-tolerance, Reliability, Security, and Survivability”, IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 11, NO. 2, SECOND QUARTER 2009
[49] T. Charles Clancy, Nathan Goergen, “Security in Cognitive Radio Networks: Threats and Mitigation”
[50] S.C.Lingareddy et al., “Wireless Information Security Based on Cognitive Approaches”, IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.12, pp. 49-54, December 2009
[51] Jan Peleska, “Formal Methods and the Development of Dependable Systems”\
[52] Martin ARNDT, “Towards a pan European architecture for cooperative systems”, ETSI Status on Standardization, 2009
[53] AbdelNasir Alshamsi, Takamichi Saito, “A Technical Comparison of IPSec and SSL”, Tokyo University of Technology
[54] Your Electronics Open Source, “Embedded Systems in SDR and Cognitive Radio”, http://dev.emcelettronica.com/
[55] V. Casola, A. Gaglione, and A. Mazzeo, A reference architecture for sensor networks integration and management, 2009. In IEEE Proceedings of GSN09, Oxford, July 2009
[56] G. Wiederhold, Mediators in the Architecture of Future Information Systems, In IEEE Computer XXV(3), pp. 38-49, 1992
[57] Kirk Martinez, Jane K. Hart, and Royan Ong. Environmental sensor networks. Computer, 37(8):50–56, 2004
[58] B. Warneke, M. Last, B. Liebowitz, and K.S.J. Pister. Smart dust: communicating with a cubic-millimeter computer. Computer, 34(1):44–51, Jan 2001
[59] V. Casola, A. De Benedictis, A. Mazzeo and N. Mazzocca, SeNsIM-SEC: security in heterogeneous sensor networks, May 2011, SARSSI2011
[60] TinyOS Project, URL: http://www.tinyos.net
[61] H.Wang, B. Sheng, C.C. Tan and Qun Li, WM-ECC: an Elliptic Curve Cryptograph Suite on Sensor Motes, Technical report, Oct. 30, 2007

[62] Damasio A. (2000), “The feeling of what happens - body, emotion and the rise of consciousness”, Harvest Books
[63] Brdiczka O., P.C. Chen, S. Zaidenberg, P. Reignier and J.L. Crowley (2006), “Automatic Acquisition of Context Models and its Application to Video Surveillance”, International Conference in Pattern Recognition, 1175-1178, DOI:10.1109/ICPR.2006.292
[64] Marcenaro L., Oberti F., Foresti G., Regazzoni C. (2001), “Distributed architectures and logical-task decomposition in multimedia surveillance systems”, Proceedings of the IEEE (10) (October 2001) 1419-1440
[65] Moncrieff S., S. Venkatesh e G. West (2008), “Context aware privacy in visual surveillance”, International Conference in Pattern Recognition, 1-4, DOI:10.1109/ICPR.2008.4761616
[66] Remagnino P. e G.L. FORESTI (2005) Ambient Intelligence: A New Multidisciplinary Paradigm, Machine Vision and Applications, IEEE Transactions on Systems, Man and Cybernetics - Part A, 35, 1-6, DOI:10.1109/TSMCA.2004.838456
[67] Velastin S., L. Khoudour, B.P.L. Lo, J. Sun and M.A. Vicensio-Silve (2004) Prismatica: A multi-sensor surveillance system for public transport network, 12th IEE Road Transport Information and Control Conference, 19-25
[68] Sarfraz Alam, Josef Noll, “Enabling Sensor as Virtual Services through Lightweight Sensor Description”, in the proceeding of fourth International Conference on Sensor Technologies and Applications (SENSORCOMM 2010), July 18 - 25, 2010 - Venice/Mestre, Italy, pp. 564-569, ISBN: 978-0-7695-4096-2
[69] Sensor model language (SensorML). (2005)[online], http://www.opengeospatial.org/standards/sensorml
[70] B. Adida, M. Birbeck (2008). RDFa primer: Bridging the human and data webs. [online] available: http://www.w3.org/TR/xhtml-rfda-primer
[71] R. Khare (2006), Microformats: The next (small) thing on the semantic web? Internet Computing 10(1), 68-75
[72] B. Ostermaier, K. Romer, F. Mattern, M. Fahrmair, & W. kellerer (2010). A real-time search engine for the Web of Things. Proceedings of Internet of Things 2010, pp. 1-8