Project no: 269317

nSHIELD

new embedded Systems arcHItecturE for multi-Layer Dependable solutions

Instrument type: Collaborative Project, JTI-CP-ARTEMIS

Priority name: Embedded Systems

D2.4: Reference system architecture design

Due date of deliverable: M12 – 2012.08.31

Actual submission date: M13 – 2012.09.30

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

Organisation name of lead contractor for this deliverable:

Hellenic Aerospace Industry, HAI

Revision [Issue 11]

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

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		
Spase Drakul	THYIA		
Ljiljana Mijic	THYIA		
Andreas Papalambrou	ATHENA		
Kresimir.Dabcevic	UNIGE		
Luigi Trono	SG		
Reviewed by			
Name	**Company**		
Elisabetta Campaiola	SE		
Approved by			
Name	**Company**		
Luigi Trono	SG		

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

Modification History		
Issue	**Date**	**Description**
Issue 1	27.08.2012	1st issue based on D2.3 “Preliminary system architecture design contents
Issue 2	01.09.2012	Incorporation of comments and material that was initially omitted in D2.3
Issue 3	05.09.2012	deployment view – node layer (ACORDE) added, deployment view – network layer (HAI) updated
Issue 4	14.09.2012	Section 6.1 updates, Section 6.3 updates
Issue 5	16.09.2012	Modified introduction of section 6
Issue 6	19.09.2012	Updates of section IDS service in section 6.3
Issue 7	20.09.2012	Added scenario realization in section 8, Update of Node Layer
Issue 8	25.09.2012	Update of section 8, minor improvements in sections 6.2, 6.3
Issue 9	27.09.2012	Update of sections 6.3 (Smart SPD Transmission, Location Anonymity) and section 7.3 (component interfaces)
Issue 10	28.09.2012	Update of sections 6.4 (Middleware layer)
Issue 11	30.09.2012	Update of sections 6.5 (Overlay layer), Update of tables 7.1, 7.2, finalization

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

D2.4 is the second of the three deliverables concerning nSHIELD System Architecture (the previous was D2.3 and the final one is D2.7) inside nSHIELD WP2, “Scenarios, requirements and system design”. It is a public deliverable and as denoted by its title, the main objective is to define the Reference nSHIELD System Architecture. According to nSHIELD DoW’s specifications, taking over from D2.3, D2.4 introduces 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 envisaged architecture should:

- Be coherent, composable and modular in order to allow for a flexible distribution of SPD information and functionalities between different ESs while supporting security and dependability characteristics
- Be adaptable to each defined application scenario and cover most of their SPD requirements
- Ensure the correct communication among the different SPD modules

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 the functional system architecture is one of the most critical project tasks.

nSHIELD System 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 implemented by SPD modules are:

- At node layer, providing SPD intrinsic capabilities through the creation of an Intelligent ES HW/SW platform consisting of three different kinds of Intelligent ES Nodes
- At network layer, improved SPD technologies, secure, trusted, dependable and efficient data transfer based on self-configurations, self-management, self-supervision and self-recovery
- At middleware and overlay layer,
    - semantic technologies to address the interoperability among different SPD technologies
    - SPD core services to achieve service management functionalities (secure service discovery and delivery, service compositions, measurement systems and data distribution systems) and context awareness features
    - policy-based management to ensure high level of SPD of a system composed of Intelligent ES nodes
    - adaptation of legacy systems to ensure use of legacy nodes and networks,
    - overlay monitoring by security agents to ensure intelligent, adaptive, autonomous and cooperative capabilities
    - composability of SPD modules belonging to all the three layers (node, network, middleware)
    - secure and efficient resource management, inter-operation among heterogeneous networks

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 rather 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.

In chapter 7 of interfaces, compared to the previous version (D2.3), the definition of some interfaces was added, leaving the final analysis for the final architecture version (D2.7).

Similarly the finalization of chapter 8, including the validation of architecture through the implementation of the four predefined application scenarios, is transferred for the next deliverable version, whereas here the outline of first scenario's (Railway Security) realization is illustrated.

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 [11]. 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" [12] that makes it easier for software developers to perform communication and input/output, by hiding operational system's details from the application developers, so they can focus on the specific purpose of their application.

According to [11] 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 System 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 devices as well as as with the external world.
- **Middleware Layer** includes the software functionalities that enable:
  - basic nSHIELD services to utilize underlying networks of embedded systems (like service discovery and composition)
  - basic nSHIELD 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, these properties (i.e. energy) cannot be regarded to be part of a single layer but rather they impose cross-layer constraints and requirements.

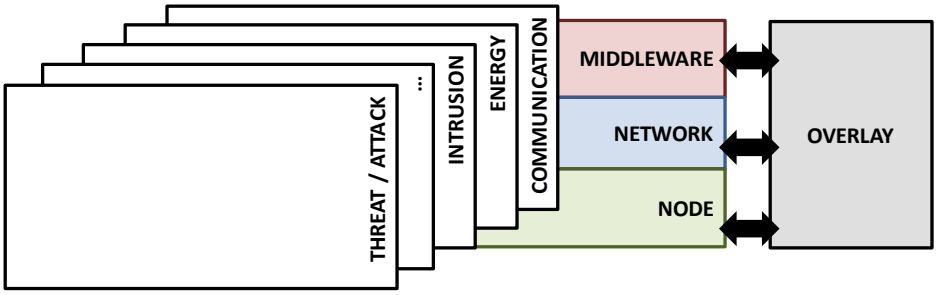

Figure 3-1: The four functional layers of an nSHIELD system

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 [18] 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 system architecture. The present deliverable (D2.4) 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 system architecture based on additional feedback from involved stakeholders, 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 D2.7) 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 consideration what is possible and look at the requirements from a risk/cost perspective.

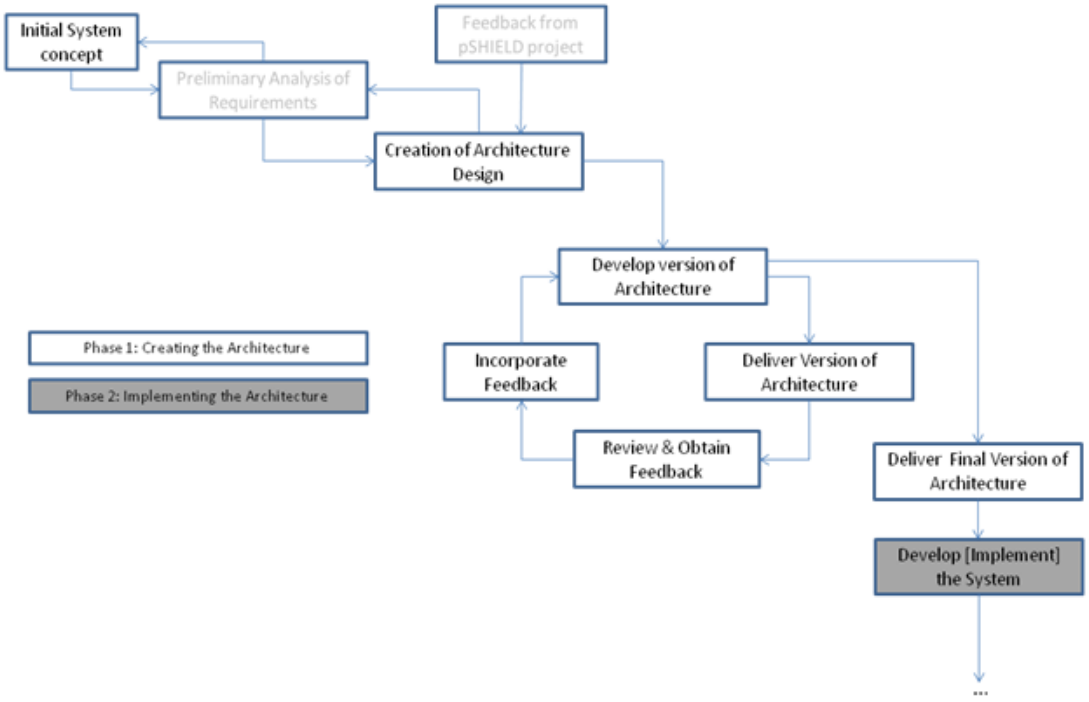

Figure 4-1: Process of Architecture definition in the nSHIELD case

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 [19] and its successor IEEE ISO/IEC 42010:2007 [20] 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 [21] and DODAF [22], developed initially for describing systems of defence domain, TOGAF [23] that targets enterprise information architectures, 4+1 Architecture View model [25] for software intensive systems, RM-ODP [24] 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 provides. 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 or structure 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 project 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 ESs. 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.

Based on pSHIELD experience and what is prescribed in the Technical Annex, the organization of the overall secure service chain architecture has been done according to three basic layers, namely the *Node Layer*, the *Network Layer* and the *Middleware Layer*. This splitting is motivated by the peculiarities of the SPD solutions to be implemented namely at node (hardware and firmware), network (protocol and data transfer) and middleware (software) level and by the actual industrial contest of embedded system suppliers, which are mainly organized in these three major sectors. An additional *Overlay Layer* is foreseen for collecting and storing semantic information from the other 3 layers (see also **nSHIELD System Architecture** definition in Terms and Definitions section). Its purpose will become more evident after section 0. The nSHIELD architecture should be also generic enough in order to allow participation of legacy embedded systems not capable to support the nSHIELD SPD modules.

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 be executed 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 encapsulates the complexities and privacy issues that may arise.

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 system 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 system 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 [8] and the OSGi Service Platform [5], while there are other approaches like JINI services [7] or Open Grid Service Architecture (OGSA) [6]. More traditional approaches include the Common Object Request Broker Architecture (CORBA) [13] and Microsoft’s proprietary Distributed Component Object Model (DCOM) [14]. Although the majority of them were initially developed to address general purpose computer systems, their basic principles could possibly 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 specifications 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 [9] [10]. 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 [15]:

- **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 necessarily 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)**: devices with increasing complexity 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 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 [16] and Prism-MW [17]. 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 in deliverable D2.2 [3] divides nSHIELD system requirements in two main classes:

(Class A): nSHIELD high level requirements

(Class B): nSHIELD layers requirements

The first class contains requirements that pertain to an nSHIELD system in general. It includes requirements that relate to both functional and SPD features that must be supported by an embedded device or network of devices. Although they are quite high level they provide a first context of the needs for defining the nSHIELD architectural framework. These requirements are REQ_SH1 – REQ_SH30. There are also a number of non-functional requirements, namely REQ_SH31 – REQ_SH35, that deal with quality engineering issues and which are of no interest for architecture definition but rather have to do with quality control.

Class B requirements are subdivided based on the nSHIELD four (4) functional layers. In some cases these requirements can be regarded as refinements over Class A requirements. In the present deliverable, the analysis of Class B requirements was the major input source for proceeding with a more precise definition of “services” and SPD capabilities for each layer. They were also used to drive the identification of components and interactions that implement the various views included throughout sections 6.2 – 6.5. The architecture analysis performed in the following part of the document tries to addresses at least all requirements defined as SHALL in D2.2 plus some requirements considered under the SHOULD or MAY keywords.

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 System 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 System Architecture

This chapter focuses on providing detailed information regarding the nSHIELD system architecture. The nSHIELD system 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 system.

In the present version of the nSHIELD system architecture document, we elaborate initially the logical view for each functional layer. This view provides an overview of the major (SPD) services potentially supported by each layer. In our analysis context, the term service, wherever it is used, should be regarded to have broader meaning than the one its normal definition implies. A service is expected to encompass all the needed functionality (including components, resources, etc.) that are needed to implement/provide a certain SPD capability or a group of them. For each functional layer there is also a separate section devoted to the development and deployment view. These two views were considered together since they are based on similar types of diagrams. Sometimes it was difficult to distinguish between them and in the high level analysis that we performed they contain quite overlapping information. In the final version of the document (D2.7) and after the demonstrators are fully defined, the deployment view (physical view) might be enhanced and fully separated.

The process view is not yet provided. Certain dynamic aspects of the system are still under consideration and many behavioural features will not be stabilized before work in WP3 (node), WP4 (network), WP5 (middleware & overlay) has proceeded to a certain level. In many cases describing the system processes involves interactions between components residing in different nSHIELD functional layers and this adds a level of complexity which cannot yet be addressed. The final version of the system architecture (deliverable D2.7) is expected to contain the process view as well as possible updates to the other views.

It must be noted that the SPD capabilities prescribed in sections 6.2 to 6.5 are the potential ones we would like to see implemented. Since we define generic reference architecture the aim was to address all possible SPD related services that should be supported by an “ideal” nSHIELD ESD. However in a real world scenario it is less than probable that all prescribed capabilities, especially at node and network layer, would be present in every device and available for discovery and composability by higher layers. Therefore a realization of the reference architecture should be performed for each nSHIELD application scenario providing among others a list of the expected services to be supported. This is the intention of chapter 0.

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 resource 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 these, 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.
- ***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.

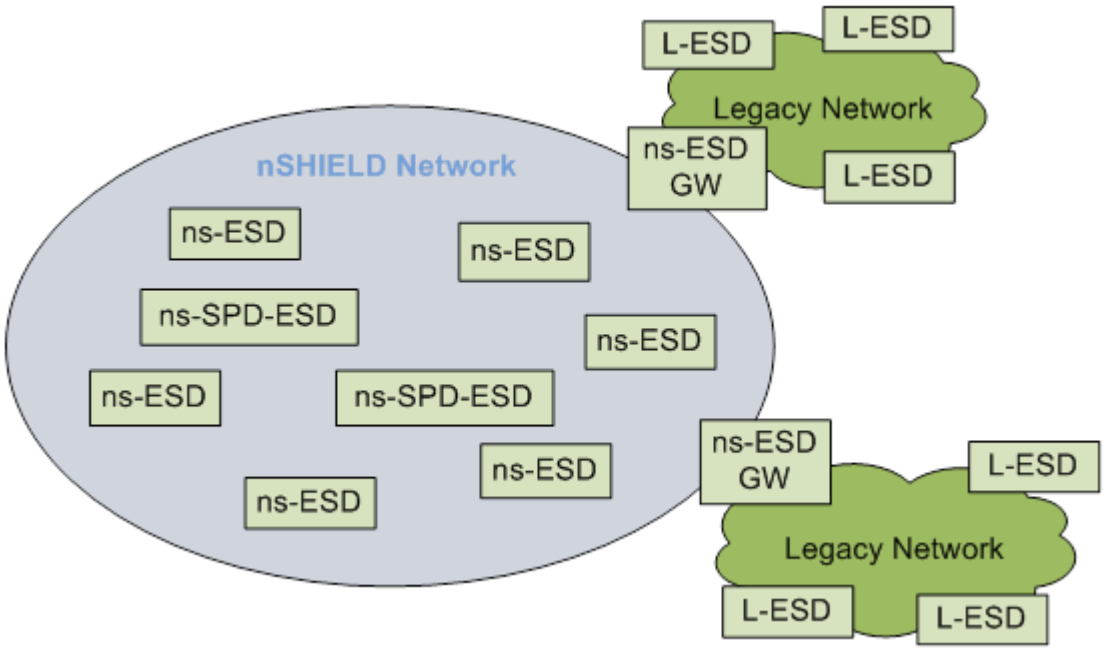

Figure 6-1: Conceptual Architecture of the nSHIELD system (Physical view)

[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
- *Adapter* 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

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 elements. The conceptual architecture of an nSHIELD subsystem is depicted in Figure 6-3. The "nSHIELD Middleware/Network" cloud abstracts all the network infrastructure and basic services supported by the nSHIELD middleware.

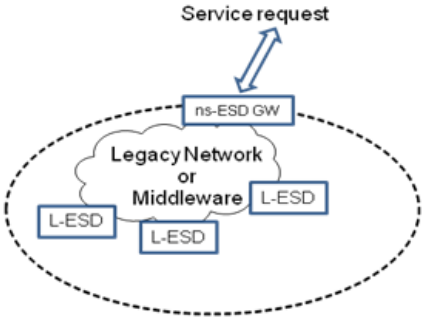

Figure 6-2: Architecture of an nSHIELD aware cluster

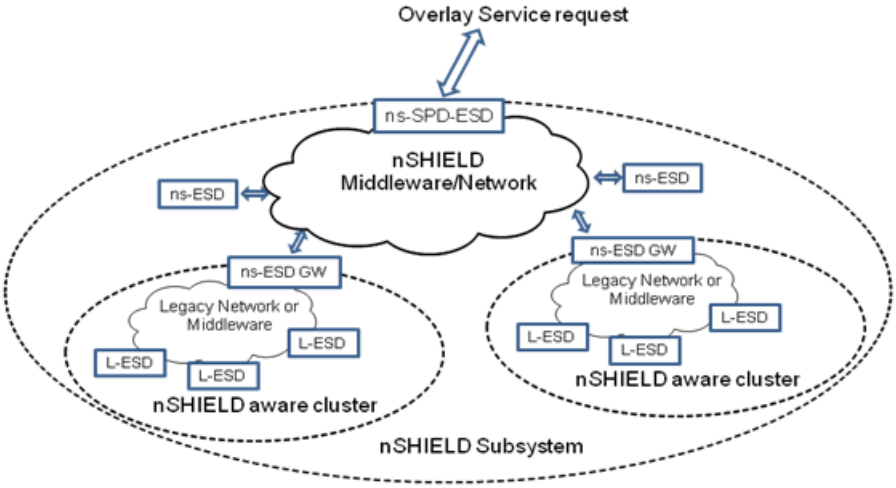

Figure 6-3: Architecture of an nSHIELD subsystem

[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.

Putting everything together the overall conceptual architecture of an nSHIELD system is illustrated in Figure 6-4 . The “nSHIELD Overlay” cloud abstracts 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 an 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.

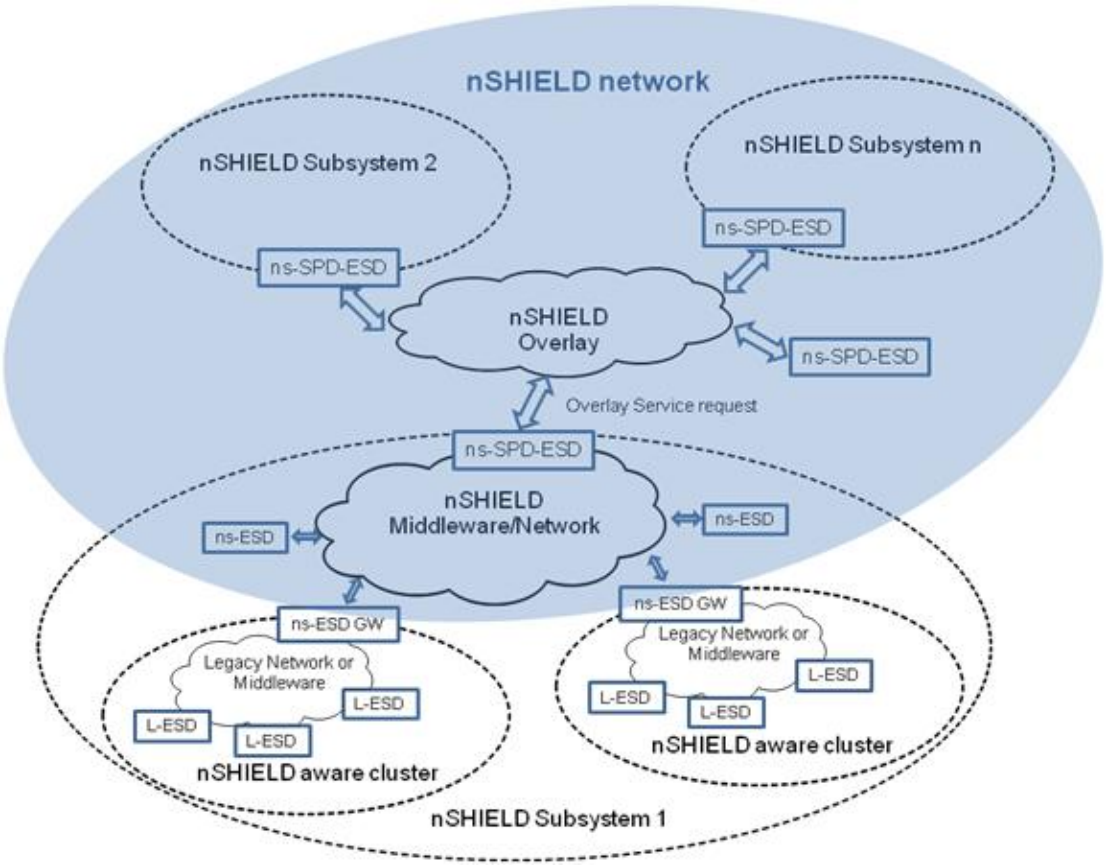

Figure 6-4: Conceptual Architecture of nSHIELD System (Hierarchical logical view)

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 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 chapter 0 which focuses on the interfaces while a detailed analysis of the depicted functional layers will be given in the sections that follow. From the figure it is evident that any nS-ESD can be transformed to nS-SPD-ESD if overlay services can be deployed on it.

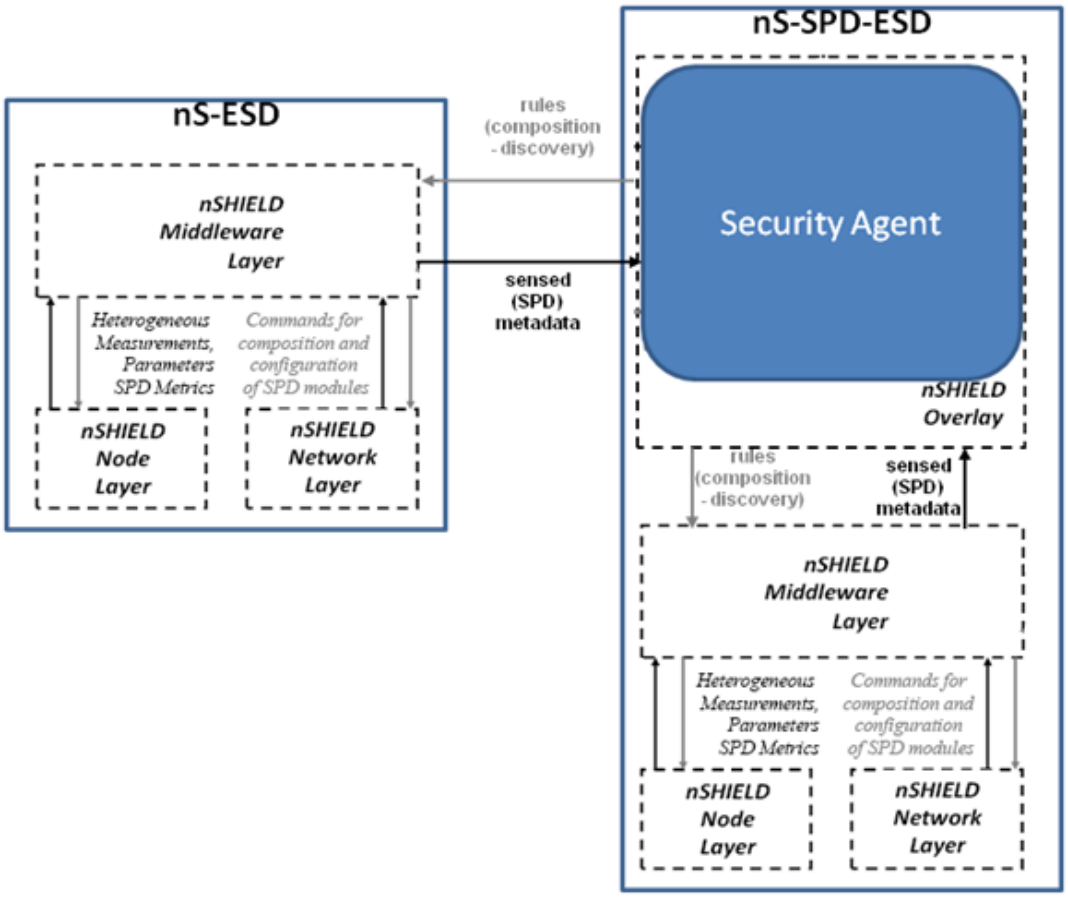

Figure 6-5: Internal architecture of nSHIELD ESDs with respect to the 4 functional layers

In Figure 6-6 we give a simplified internal structure of the nS-ESD GW. As already stated this node may contain additional technology dependant components which may either translate nSHIELD middleware requests to the ones that a legacy middleware can process or adapt proprietary network interfaces so that interaction within the nSHIELD network is possible for non nSHIELD compliant devices. Besides that the normal nSHIELD layer stack is present. Since the overlay layer is completely independent from the proxy component given in Figure 6-6, nothing prevents an nS-SPD-ESD to also act as a gateway for legacy devices provided that the proxy module is implemented on it.

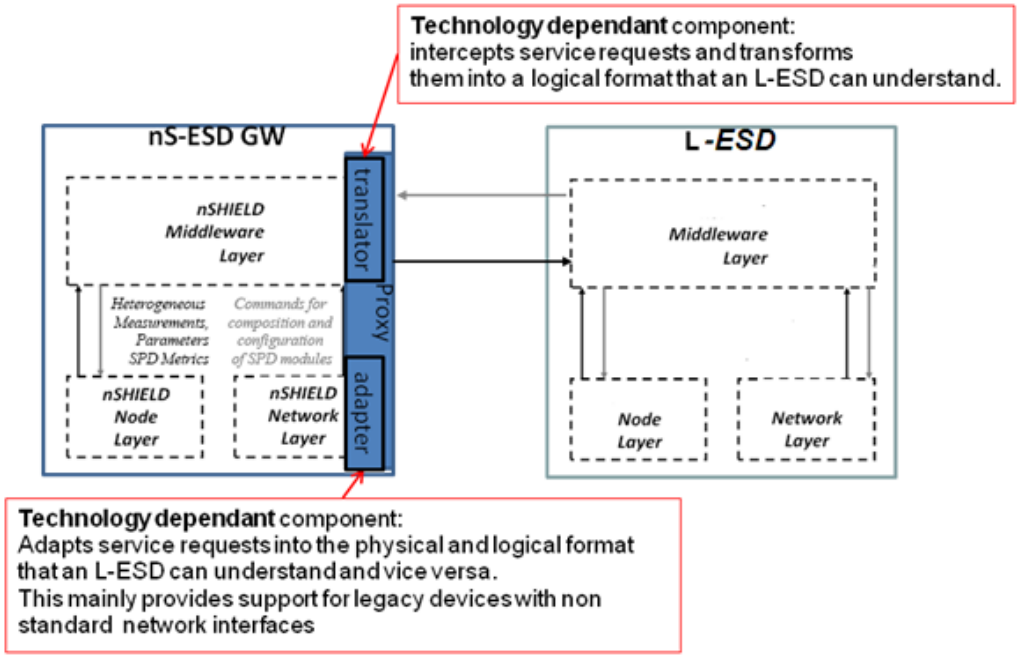

Figure 6-6: Internal architecture of nSHIELD ESD GW depicting the technology dependant components

6.2 Node

In a previous deliverable [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 features 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.

The **Node Layer** includes the hardware components that constitute the physical part of the system and the basic SPDs capabilities that can be implemented at OS/HAL level (this does not cover the advanced SPDs ones that would be implemented in upper layers).

The full definition of the node layer depends on the final scenario where the node will be used and the required capabilities for that node. Therefore, in this section what is proposed is a set of basic capabilities at node layer that will support the final composition of capabilities in all nSHIELD layers and prove that a desired SPD attribute level is achieved within each application scenario.

6.2.1 Logical View and Modules Description

Figure 6-7 provides a logical view of all possible modules while more details on them are provided in the next paragraphs.

The logical view has been modified (with respect to previous version) in order to provide a better overview of the capabilities provided by the Node layer. Capabilities are implemented by specific components that are grouped into higher level modules. Thus, modules are conformed by components with related features or dedicated to the same area.

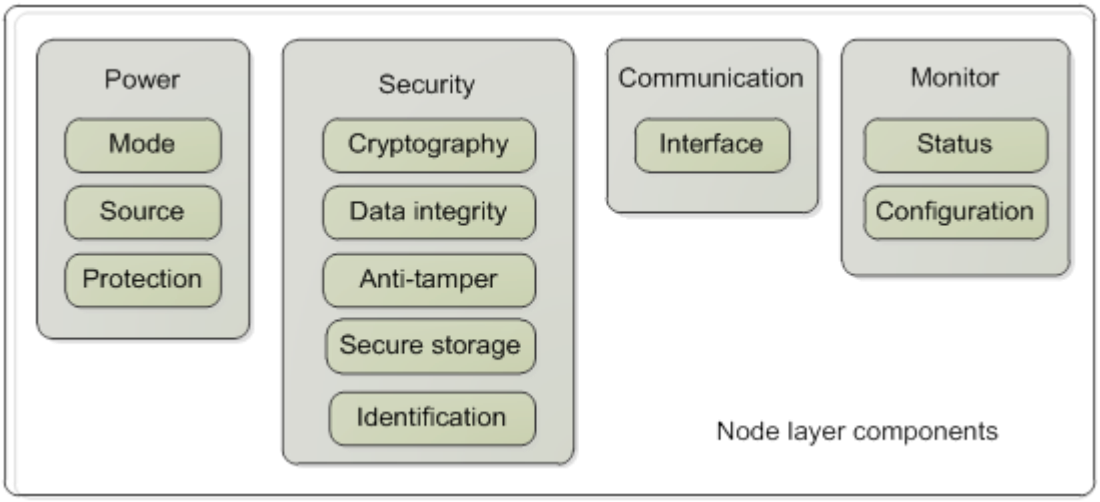

Figure 6-7: nSHIELD's Node Layer Modules (Logical View)

Four main modules are proposed. Each module consists on a set of several components which implements specific capabilities. Each component can be HW, SW or, commonly, a mixed HW/SW solution. An example of component could be a chipset and its associated low-level driver that abstracts the hardware details to the upper layer.

Depending on the type and complexity, an nSHIELD node could implement some or none of the components described for any of these modules. Simple nSHIELD nodes could just implement a very basic subset of components whereas an advanced node could implement many of them. In most cases it may be unnecessary to implement all the components.

The following sections describe each module and its components. Also, it is presented a list of attributes for every component in the node layer. Depending on the kind of nSHIELD node, some of these attributes could be implemented or not. These attributes will support the SPD attributes and the metrics defined in the scope of an nSHIELD device.

6.2.1.1 Power module

This module may implement some of the following components and attributes:

Table 6-1: Power module at node layer

		Attributes				
Power module	**Mode**	Low power				
	Source	Power line	Battery	Energy Harvest	AC	DC
	Protection	Fuse	Inverse			

6.2.1.1.1 Mode selection

In this case the nSHIELD node supports working at different (low) power modes in order to improve its overall consumption.

One application for this may be that one nSHIELD node could be maintained at low power mode except at the moment of data transmission, where a normal power mode would be activated.

There is also the option that the nSHIELD node could be remotely set to a low power mode and only set to normal power mode by petition.

6.2.1.1.2 Source selection

nSHIELD nodes could support alternative power modes, depending on the specific application, environmental conditions (e.g. vibration generator, micro-solar cells) or presence/absence of main power source and backup solutions (batteries).

This component will provide the status of the current power source supply that feeds the nSHIELD node.

- DC/AC.
- Voltage level.
- Power type:
    - Internal: use of internal batteries.
    - External: power line, external batteries or PSI.
    - Energy harvesting: the node could process energy derived from external sources (e.g., solar power, thermal energy, wind energy…) captured, and stored for small, wireless autonomous devices.

6.2.1.1.3 Protection

This component shall monitor and prevent any system power supply risk, which might affect the nSHIELD node behaviour. The module should be able to protect all the electronics and devices, in order to avoid further damages into the system.

6.2.1.2 Security module

The following table describes the attributes for the components included in the security module:

Table 6-2: Security module at node layer

		Attributes				
Security module	**Cryptography**	RNG	Key generation	AES engine	DES engine	
	Data integrity	Code execution	Secure boot	SCA protection	Secure firmware	Secure firmware upgrade
	Identification	MAC	ID	Endorsement key		
	Anti-tamper	Secure key storage				
	Secure storage	Secure data storage				

The capabilities described on this module could be implemented by different means. TPM chips, Smart Cards, dedicated secure supervisor chips or any other technology could provide the required features for the targeted functionality and security level of a specific node.

6.2.1.2.1 Cryptography component

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 may 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 may support asymmetric cryptography for low cost nodes and also should include an optimized hardware implementation for different algorithms.

6.2.1.2.2 Data integrity component

Since the node is the basic component of the nSHIELD system 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)

6.2.1.2.3 Identification

The nSHIELD node can provide a unique ID, MAC or a custom code for being used in authentication or access control functionalities implemented on upper layers.

6.2.1.2.4 Secure storage

An nSHIELD node should incorporate provisions that ensure the long term storage of sensitive user data.

6.2.1.2.5 Anti-tamper

An nSHIELD node should incorporate provisions that ensure the protection of critical data, such as the private key used by any cryptographic chip used by the node or any other sensitive information stored on the node. Other point to be covered is the secure storage of user data.

Therefore, counter-measurements against more or less complex physical attacks may be implemented depending on the targeted security level of a specific nSHIELD node.

6.2.1.3 Communication module

The communication module is in charge of the management of the peripheral used for communicating with the nSHIELD network, typically an RF device.

6.2.1.3.1 Interface module

The interface component is in charge of control and monitoring different parameters regarding communication, such as RSSI and output power level.

The following table lists all available attributes:

Table 6-3: Communications module at node layer

		Attributes		
Communication module	**Communication interface**	RSSI	Output power	

6.2.1.4 Monitor module

External layers use this module to get the capabilities of the nSHIELD node. It is also used to check the status/mode of the components on the node. This is a special module as it has internal access at the whole list of components and attributes.

This table shall be specific for each node, taking into account that each node will implement only the required functionality. The configuration row of this table shall include all components and attributes. On the status row shall be present all attributes that could be monitored (available) in the specific node.

Table 6-4: Monitor module at node layer

		Attributes				
Monitor module	Configuration	(Power) Mode	(Power) Source	(Power) Protection	(Security) Anti-tamper	…
	Status	(Power) Mode	(Power) Source	(Power) Protection	(Security) Anti-tamper	…

Configuration component shall include all attributes of the components described in previous sections, as it provides the list of capabilities of the node that will be made available to external layers.

Besides, the status of all attributes that could be monitored shall be provided by the status component.

6.2.1.4.1 Configuration

The node configuration (supported capabilities and attributes) shall be exposed to upper layers. It should be listed all the available capabilities supported by the current node.

6.2.1.4.2 Status

Once it is provided the list with the available components (attributes), the status of each component, when available, shall be made available to external layers for monitoring purposes.

6.2.2 Development and deployment view

In this section the node layer will be described from a system engineer's point-of-view (deployment view). The topology of software and hardware components on the physical layer will be depicted, as well as the physical connections between these components.

As it has been commented before, the functionality and SPD capabilities provided at node layer can differ for each ESD within the nSHIELD network. This may depend on the scenario and the function of each node. Taking this into account, example views are provided depicting possible deployments of the node layer components described previously.

Diagram Figure 6-8 is an **example** of deployment view for an RF node which only implements some of the components described before. In this case, some of these capabilities are provided by an MCU whereas some others, such as anti-tamper, cryptography or power source are provided by other chips. Even more,

some attributes could be provided by one chip, and other attributes from the same component could be provided by other chips.

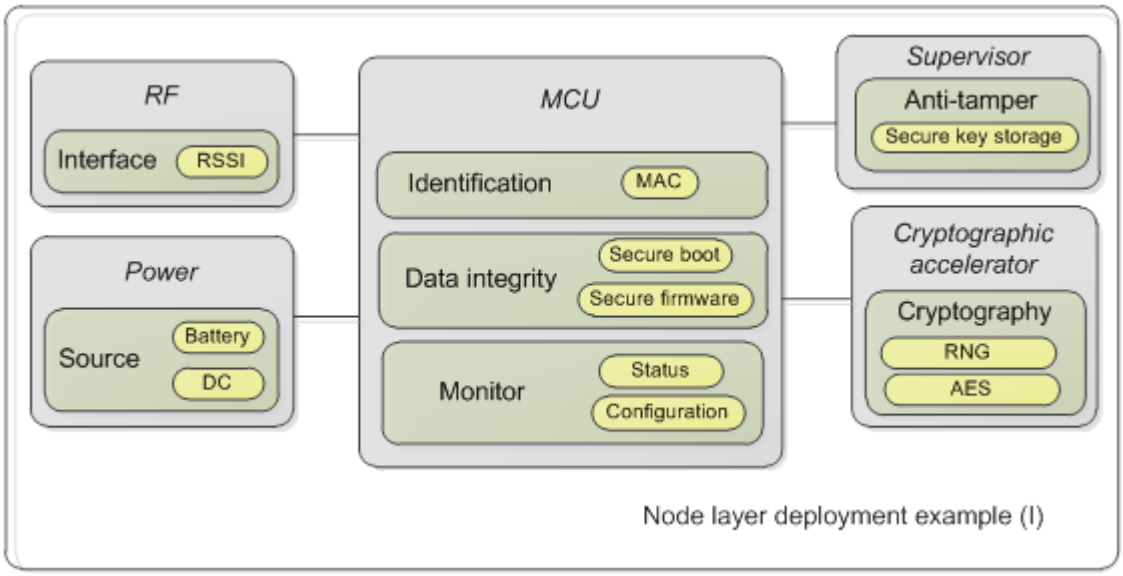

Figure 6-8: nSHIELD's Node Layer example (I)

In this example, the core of the nSHIELD node is a commercial MCU which already provides data integrity itself, without any other external requirement. However, as it does not provide any cryptographic feature, a dedicated coprocessor is used. The sensitive information path is monitored by another dedicated anti-tamper module that also incorporates a secure RAM for private key storage. This example node also provides RSSI monitoring and uses DC powered by batteries.

While in this first example the security capabilities are provided by 3 different HW elements, the example in Figure 6-9 shows how the same security capabilities can be implemented by using a single TPM chip.

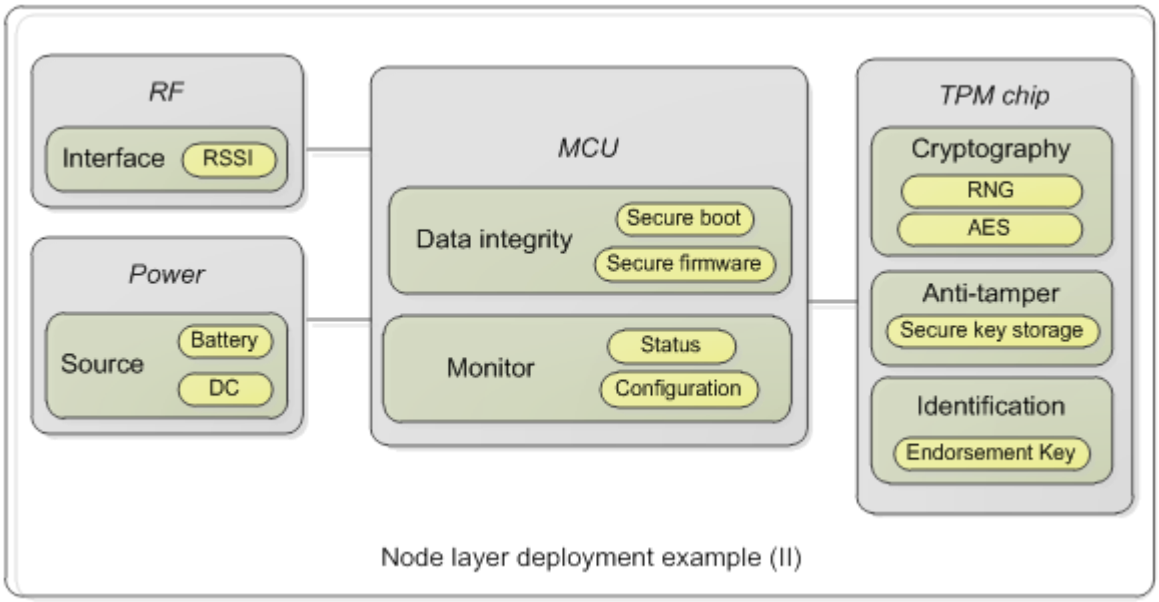

Figure 6-9: nSHIELD's Node Layer example (II)

From a development point of view, the same capabilities can be obtained with multiple HW/SW combinations. However, from a logical point of view, the node is just defined by a set of components that provide specific capabilities delimited by their corresponding attributes.

6.2.2.1 Functionalities provided to external layers

The following interfaces are mapped to the attributes that are described in the previous sections. Some attributes are static and do not provide any direct available functionality to external layers, but just reflect an intrinsic capability of the nSHIELD node (such as the power source protection or the secure booting). However, in that case, the existence/absence of this capability must be exposed to the other layers by the Monitor module.

Power module:

- Mode component:
    - Low power: GET/SET

Communications module:

- Interface component:
    - RSSI: GET
    - Output power: GET

Security module

- Cryptography component:
    - RNG: GET(generate sequence)
    - Key generation: GET(generate key)
    - AES engine: GET(decrypt)/SET(encrypt)
    - DES engine: GET(decrypt)/SET(encrypt)/
- Identification component:
    - MAC: GET
    - ID: GET
    - Endorsement key: GET
- Monitor module:
    - Configuration component (indicates if attribute exist or no)
        - Power mode: YES/NO
        - Battery: YES/NO
        - AC:YES/NO
        - …
        - RSSI: YES/NO
        - Output power: YES/NO
    - Status component:
        - Power mode: LOW POWER/HIGH POWER
        - Power protection: YES/NO
        - Battery level: FLOAT
        - Anti-tamper: SAFE/ATTACK
        - Code execution: SAFE/ATTACK

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 6-10 provides a logical view of the Network/communication Layer of nSHIELD focusing on the services that should be supported.

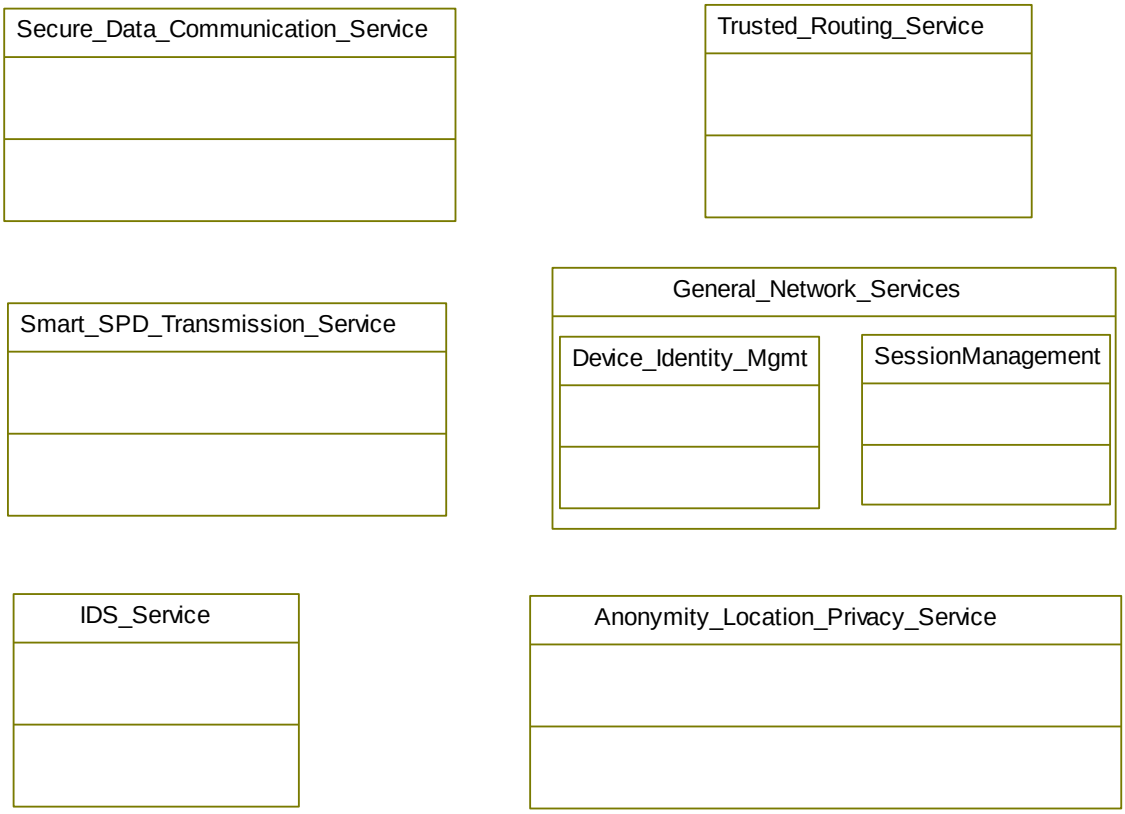

Figure 6-10: nSHIELD's Network/Communication Layer Services

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:

- 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 node 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

[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.

- 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 system 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 to the traditional network layer of the OSI model but also with the physical layer. The implementation of the smart SPD transmission service shall be based on the basic principles of Software Defined Radio (SDR) and, in instances where the node has the capability of being equipped with the more advanced awareness and learning capabilities, shall evolve towards Cognitive Radio (CR)

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.).

For establishing the maximum interoperability, the Smart SPD shall be SCA (Software Communications Architecture) compliant. Hence, in accordance with the SCA 4.0 standard, the following software components shall be present:

- Domain Manager - contains knowledge of all existing implementations installed or loaded onto the system including references to all file systems (through the file manager), device managers, application factories and applications (and their resources)
- Device Manager - contains complete knowledge of a set of devices and/or services. A system may have multiple device managers but each device manager registers with the domain manager

to assure that the domain manager has complete cognizance of the system. A device manager may have an associated file system (or file manager to support multiple file systems)

- Application Manager - provides access to a specific application that is instantiated on the system.

The scheme of these three components, along with the support for the Application Factory, File Manager and File System components, is given in the figure debajo de:

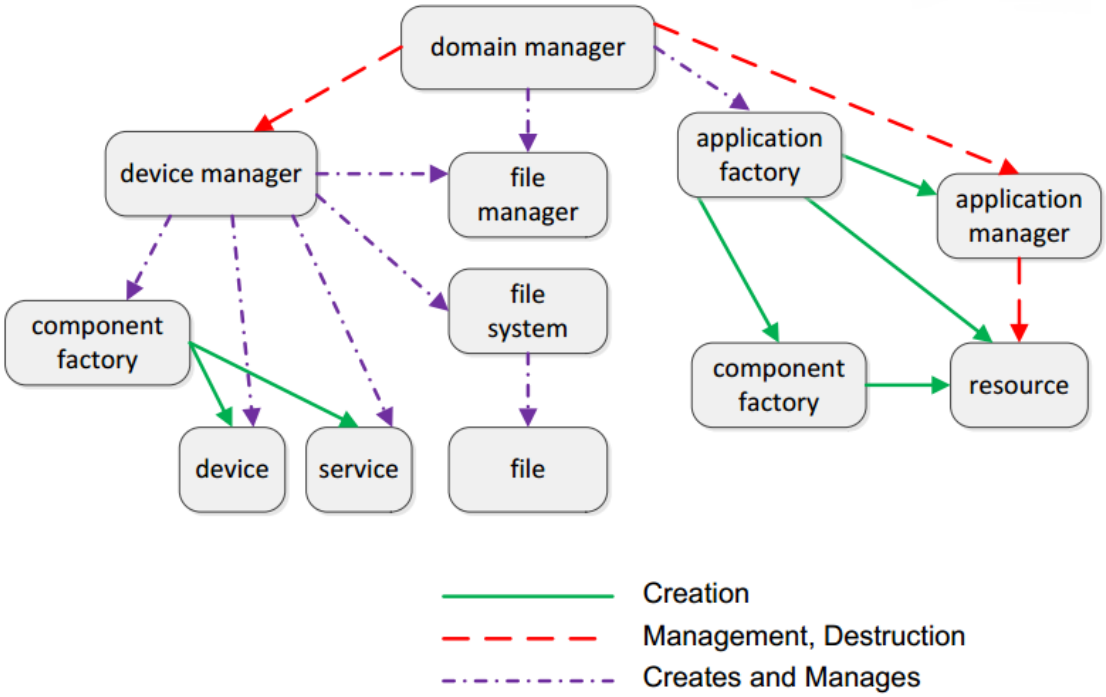

Figure 6-11: Software Communications Architecture (*source: SCA 4.0 specification [90]*)

For the purposes of ensuring the maximum security, the currently developed Security-aware framework shall be deployed. The purpose of the framework is protecting the system against the breaches and security issues that stem out from the reconfigurability characteristics of the Software Defined Radios and (in case of Cognitive Radios) the cognitive capabilities of CRs. Depending on the required level of the SPD – which shall be provided by the Overlay (or Middleware) layer – the Security-aware framework will have the capability of deploying different state-of-the-art technologies for detecting and countering the most common reconfigurability-related and cognitive capability-related security issues and attacks.

A simplified example of a scenario-dependant proposal of a Security-aware framework is shown in the figure:

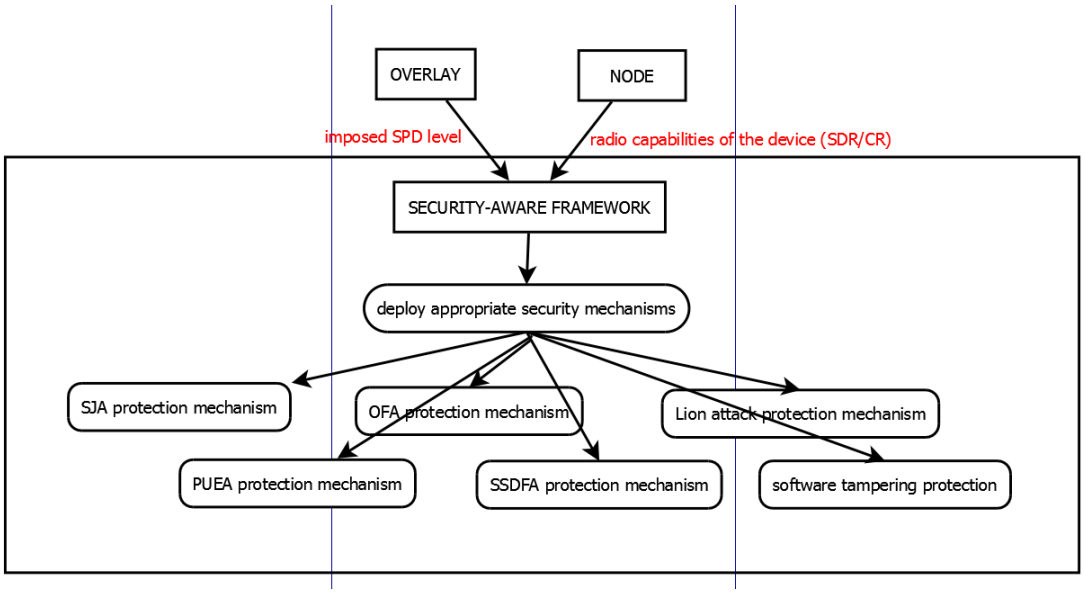

Figure 6-12: SDR Security-aware framework depicting the possible security mechanisms

The deployed mechanism depends on the SPD level imposed by the upper layers, as well as the cognitivity and reconfigurability capabilities of the device – i.e. most of the defence mechanisms depicted (against Primary User Emulation (PUEA), Smart Jamming (SJA); Spectrum Sensing Data Falsification (SSDFA); Objective Function attack (OFA; Lion attack) refer to the more advanced devices equipped with cognitive capabilities, and may be not available in simpler systems (basic SDRs).

6.3.1.3 Anonymity and location privacy service

Privacy is about protecting sensitive information such as home address, phone number, email address and other attributes, e.g. religion, political affiliations, personal activities, social relations, etc. Besides protection of sensitive data it may also include means to prevent the exposure of information related to the actual location, identity or movement of a person, device or item. This is often referred to as location privacy. An increased level of security may require that sensitive identities of certain devices should not be exposed and there is a need to randomize them so that tracking will be impossible or hard for an attacker.

Identity protection or randomization can be done in several different ways. The most common approaches include:

- Local (node) identity randomization [79]
- Through the assistance of an anonymizer proxy or server [80][81] that can either decentralise data, encrypt them, change the traffic pattern or implement a k-anonymity protocol[4] for location cloaking.
- Never exposing real identities but use cryptographic protocols which guarantee anonymity [82][83][84][85][86]

[4] The concept requires that an entity's location information are sent in such a form that make said entity indistinguishable from k-1 other neighbouring entities. Naturally, a lower k value provides less privacy protection but also less communication overhead and better quality of monitoring and the opposite stands for higher k values. The level of anonymity (k-level) required should be modifiable, depending on the context (thus dictated by the active security policy), and any such changes should be communicated to the nodes by the location-information collection point (i.e., server).

- Only send real identities in encrypted form and not in clear [87][88]

Even if measures have been taken, the communication context generally makes impossible to avoid that some location information is revealed [77]. The best that can be done for achieving location privacy is therefore to ensure that it is impossible for an attacker to distinguish between the identity of a particular unit and surrounding communication units. This is often referred to as cloaking services [76][89].

6.3.1.4 Intrusion Detection service

Intrusion detection acts a second line of defence to identify potential unauthorised attempts to exploit hidden system vulnerabilities. The intruder typically aims for unauthorised monitoring of communications, controlling nodes to alter routing decisions (sinkhole attack), and altering the messages exchanged among them. Detecting an intrusion is a challenging task in ad-hoc networks given the dynamic nature of the environment and the constant change of the environment parameters. The line that separates intrusion from legitimate use might be very thin, if totally absent.

In traditional (wired) networks intrusion detection heavily depends on the use of agents deployed on various network nodes. Similarly, the nSHIELD network layer also has to accommodate this type of functionality to gather and process the required information. Although IDS agents should typically be deployed on many nodes to be able to gather as much information as possible and have a more clear view of the network activities, such a configuration is bound to significantly downgrade system performance and exhaust limited resources of embedded systems [75].

Depending on the node's capabilities and the network configuration and policy, intrusion detection data might be processed locally for the particular node or forwarded to a centralised engine instead. In the latter case intrusion detection heavily depends on secure routing as the participating nodes have to find the means to forward the monitoring nodes' information to the console that will process them. Packets carrying intrusion information might be dropped for the same reasons that other packets do e.g. due to overloaded selfish or malicious intermediate nodes.

Reputation based schemas, already mentioned in the secure routing service, can be used to improve the figures of intrusion detection systems. They typically work against compromised nodes that start to misbehave and misuse the network resources and information. The set of attributes/metrics to be used for building a node's reputation must be carefully chosen.

6.3.1.5 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 & Deployment 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 6-13 where important dependencies between entities internal or external to the Network/communication layer are also drawn.

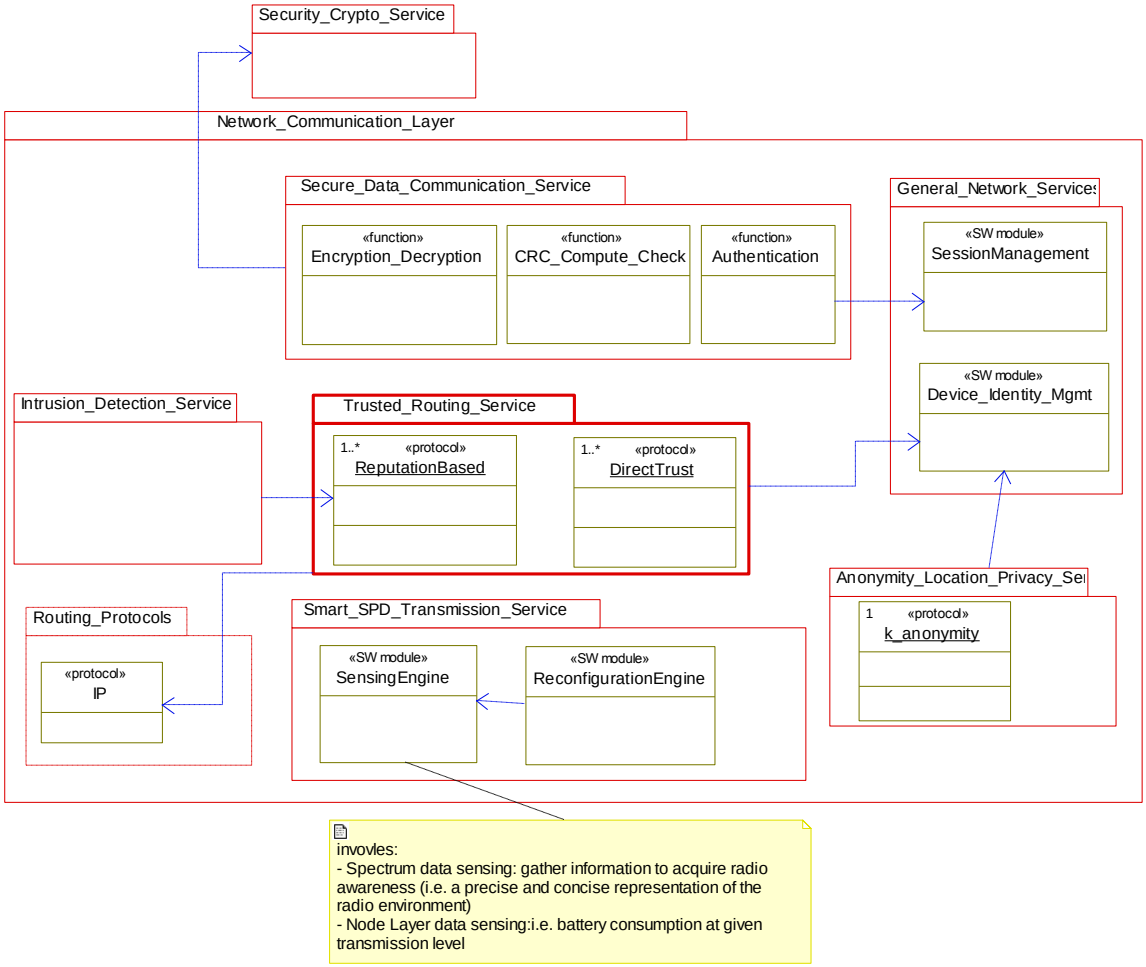

Figure 6-13: nSHIELD's Network/Communication Layer Services (Development View)

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. For each service a simplified deployment view is given seeking to provide some insight on the possible implementation of each service and the data being exchanged between the various components. This aims to act as a starting point for a more thorough analysis that should take place in WP3, which is dedicated to the actual development of Network Layer SPD capabilities.

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.

Deployment view

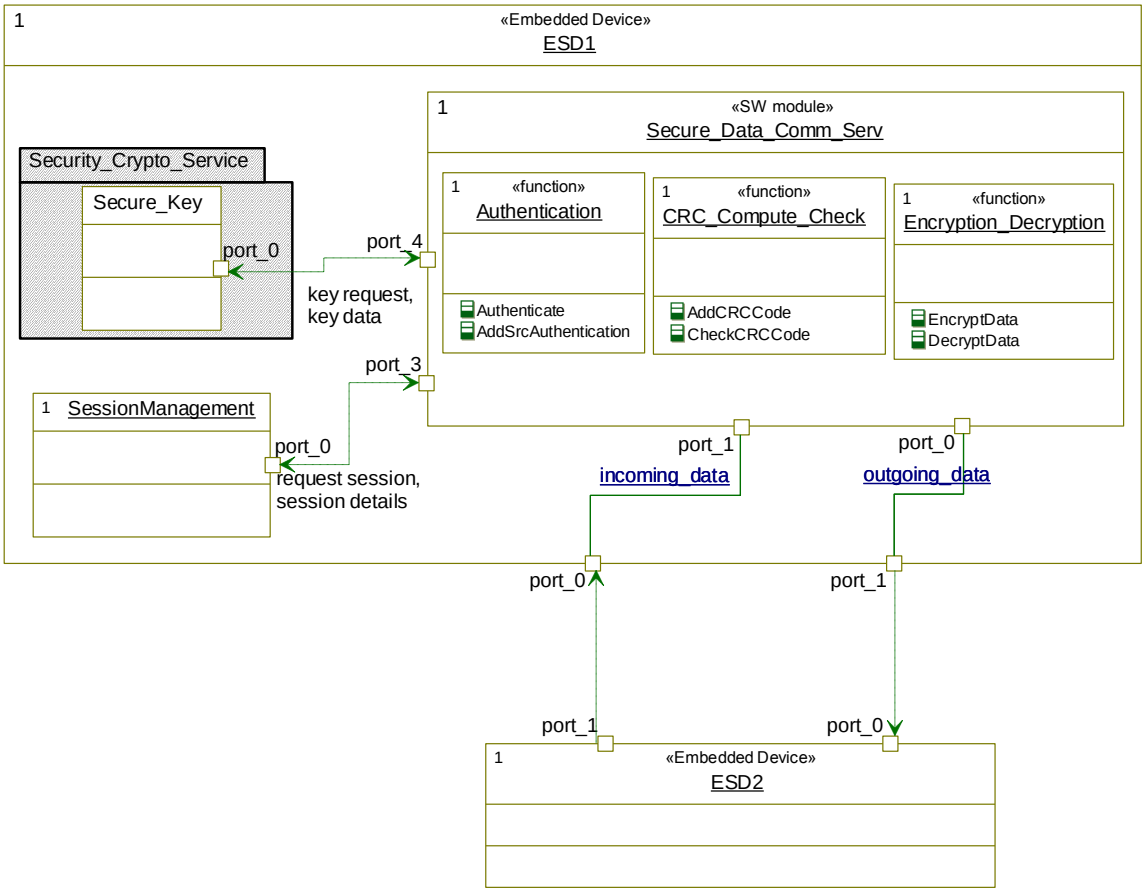

Figure 6-14: Secure Data Exchange/Communication Service (simplified deployment view)

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.

Deployment view

The diagram in Figure 6-15 depicts the deployment of the trusted routing service on an nSHIELD ESD and provides some additional information regarding the data/information exchanged between the various components implementing the secure routing functionality through a reputation based protocol.

The main units involved are:

- **Network controller**: provides configuration data to the trusted component (i.e. attributes to be considered for use in the trusted model computation). In case the ESD supports multiple trusted protocols this unit may be responsible for selecting the protocol to apply based on requests initiated by nSHIELD middleware layer.
- **Routing Protocol**: actually this component represents an existing routing protocol that runs on the node (i.e. IP).
- **Trust component:** this component is responsible for implementing a reputation based trusted model that can enhance security of a given ESD with respect to its routing functionality. The diagram also briefly analyses the component's internal structure and input and output flows.

The expected behaviour has as follows: Each node monitors events (**monitoring** module which acts as a kind of watchdog) which are then stored by the **trust manager** module component to the **direct trust** table. To evaluate the trustworthiness of a certain node, indirect information is required and is obtained through the following sequence of actions: a "reputation request" is broadcasted and the received responses are delivered to the **trust manager** module through the **monitoring** module. The content of the reputation response is stored in the **indirect trust** table, based on certain preconditions that must be fulfilled. The **trust manager** is also responsible for evaluating the aggregated trust value. Therefore it combines the direct and indirect trust information to calculate a single trust value per neighbouring node. Based on that, the **trust manager** may decide to proceed with a routing info update affecting the routing tables of the underlined routing protocol (i.e. IP). **Trust manager** is also triggered when the node is asked to provide its trust value for a neighbour, i.e. to form a "reputation response" message. Alarms are generated in conditions that may indicate a security breach and should be collected by other nodes.

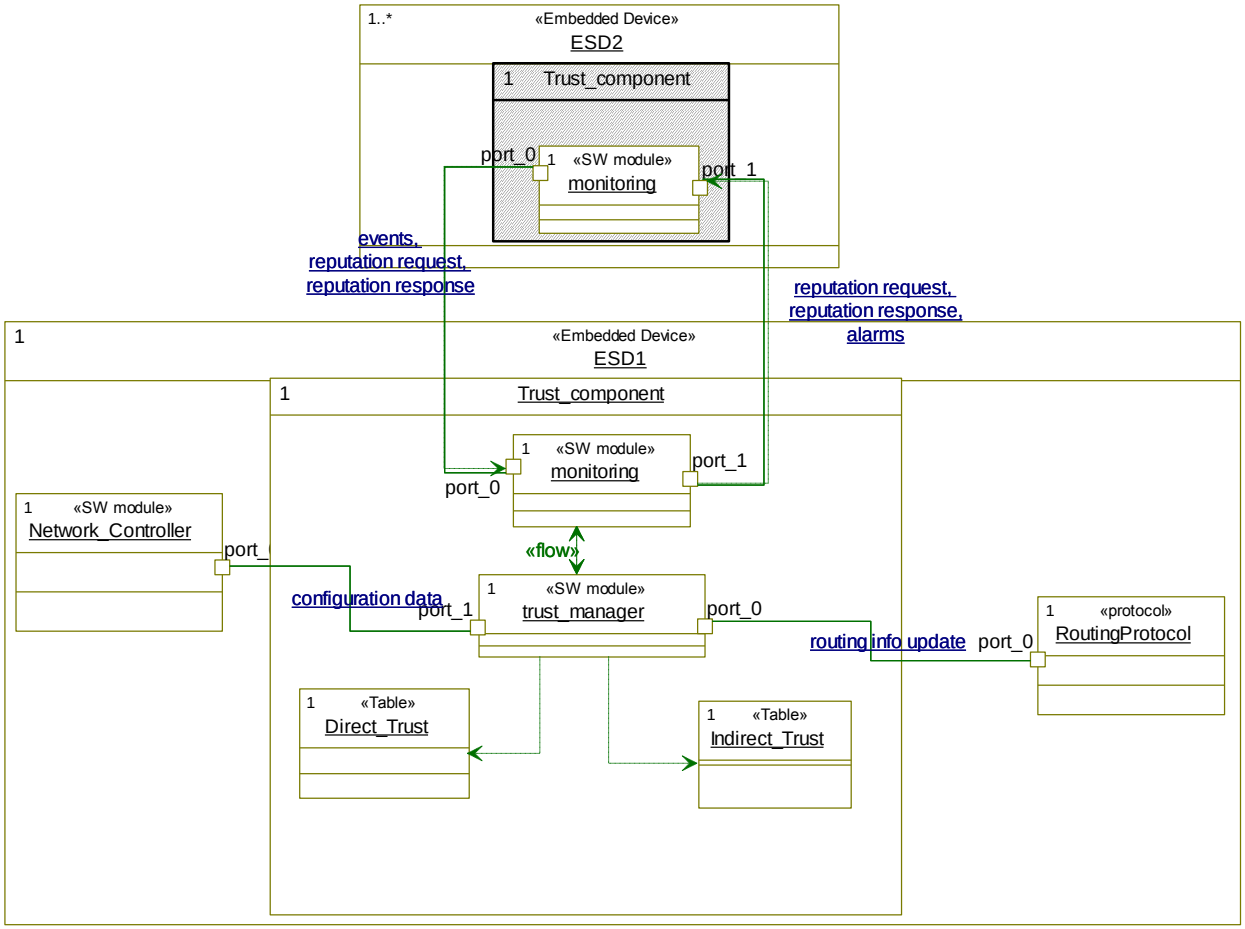

Figure 6-15: Trusted Network Routing Service (simplified deployment view)

Smart SPD transmission

The service assumes the existence of the following 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 contain interfaces with appropriate node layer services belonging to the power, communication and monitor node modules. 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. Typically, the basic parameters requested will be about power consumption, radio spectrum status and local node resources. These will be both about detecting possible attacks as well as optimizing communication in typical situations.
- A *reconfiguration/parameter adjustment module* that can utilize the information gathered above and take decisions 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. Typically, the reconfiguration commands issued will be about transmission power, frequency band used, spectrum modulation, as well as, transmission timing (wait, hold, cancel requests etc.).
- (*optionally*) A *learning module* based on one or more of the machine learning techniques, enabling the device to learn from its previous behaviours and decisions, in order to improve.

Dependencies:

The three modules may have inter-dependencies.

Regarding external dependencies, both *sensing* and *reconfiguration* modules need to interface with node layer components and in particular the monitor, communication and power modules as well as the *Communication Control and Security Policy module* (the middleware "service" that is responsible for handling communication of contracts, control commands and policies to other functional layers) of the middleware layer.

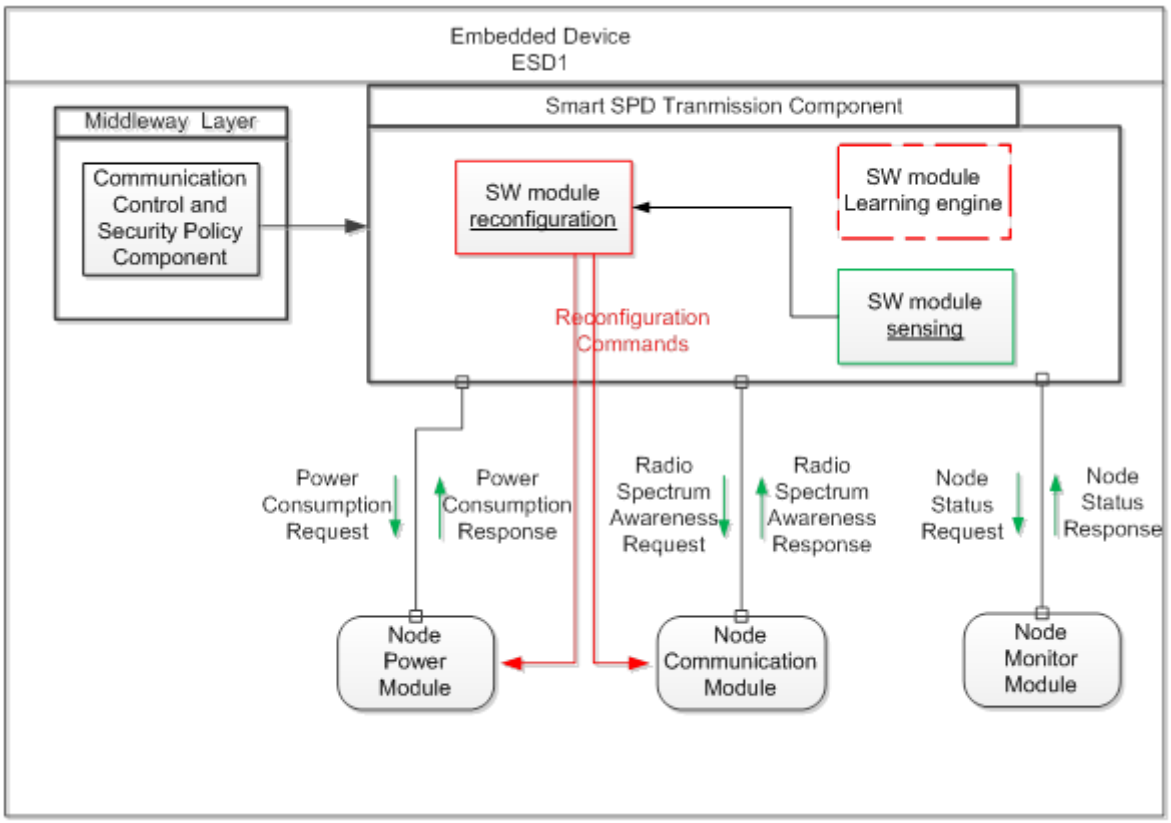

Figure 6-16: Trusted Network Routing Service (simplified deployment view)

Intrusion Detection Service

The network layer should have the capability to collect and possibly audit data to identify potential intrusions. The implementation can be based on a local (Figure 6-17) or centralized/global (Figure 6-18) detection engine scheme. In both cases a reputation based mechanism is usually employed. The local engine needs to interact with the trusted manager while in both cases (centralized and local one) a kind of watchdog is needed to gather information from the network and the local node, a functionality that can be provided by enhancing the monitoring component of trusted routing service.

Some of the above components might prove very expensive in terms of resources, hence cannot be deployed in every component. A global detection engine is typically a component expected to exist on high power nodes like nS-SPD-ESDs.

In the nSHIELD system, the functionality supported by a particular node, should be disseminated to other participating nodes through service discovery and composition mechanisms. This will allow the dynamic formulation of the infrastructure in terms of data collection and processing and the IDS adaptation to different environments based on the participating nodes capabilities. For example, some nodes might

simply provide network traffic and/or node activity logs, or might also be capable of doing some processing of those data.

Dependencies:

There might be a need to cooperate with an event logging system working on the node layer.

Components of the *Trusted Network Routing Service* are needed for monitoring and storing needed information.

Deployment view

In the following a simplified deployment diagram is provided for both the centralized and local detection case where it is evident the dependence on the trusted module of ***Trusted Network Routing*** service. The latter is of vital importance as its role is two-fold: Firstly, it will ensure that the various IDS-related messages are communicated securely to the involved nodes across the network. Secondly, information from secure routing can also be used for determining whether an intrusion has commenced (or is still in progress) or not.

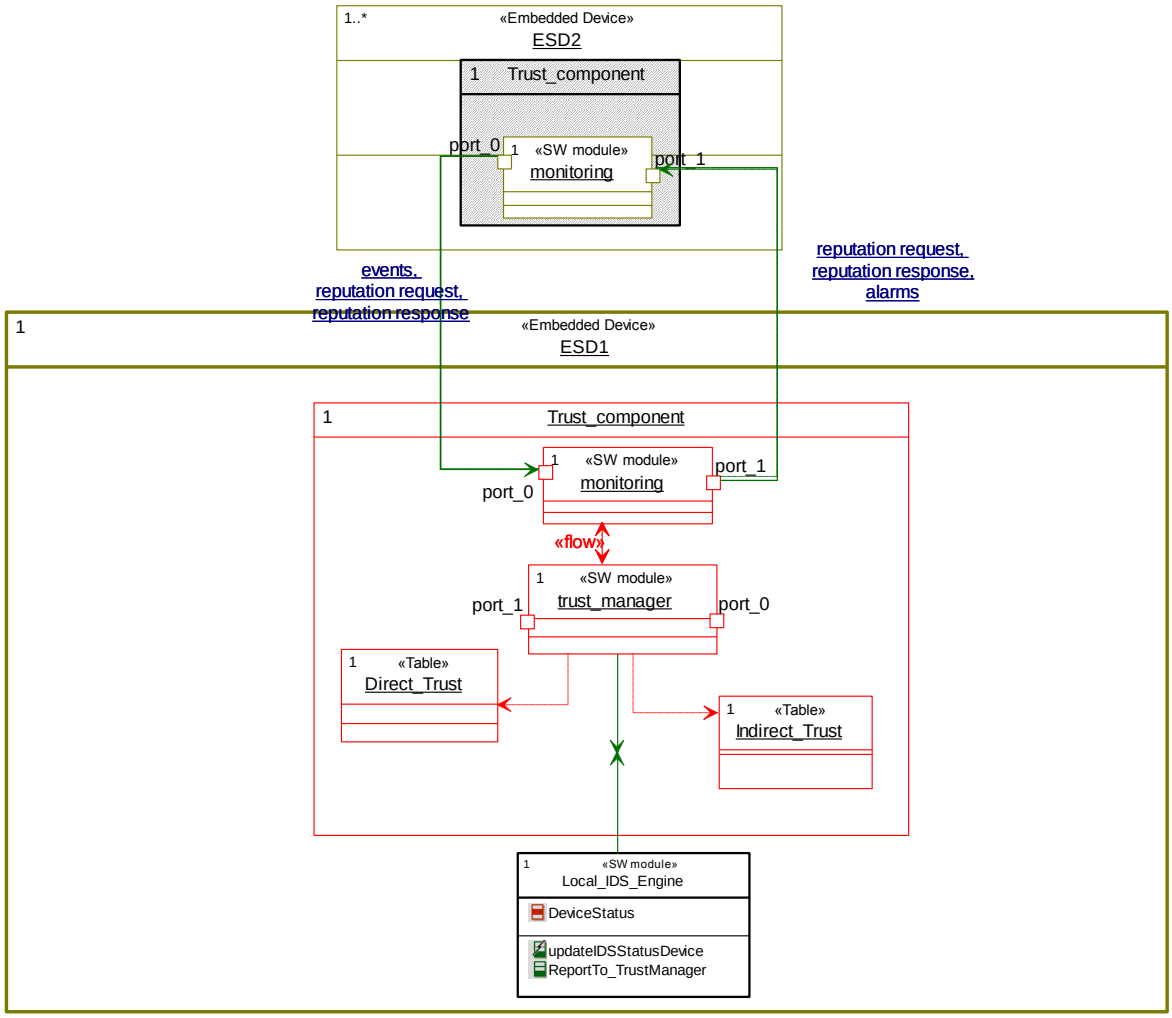

Figure 6-17: Local IDS (simplified deployment view)

In the local approach, as depicted in Figure 6-17, the trust manager establishes the various trust levels of both direct and indirect trust, and communicates these with the monitored nodes, through reputation requests/responses. Should there any abnormal behaviour be detected, an alarm is raised and the relevant message can then be sent to all the concerned nodes.

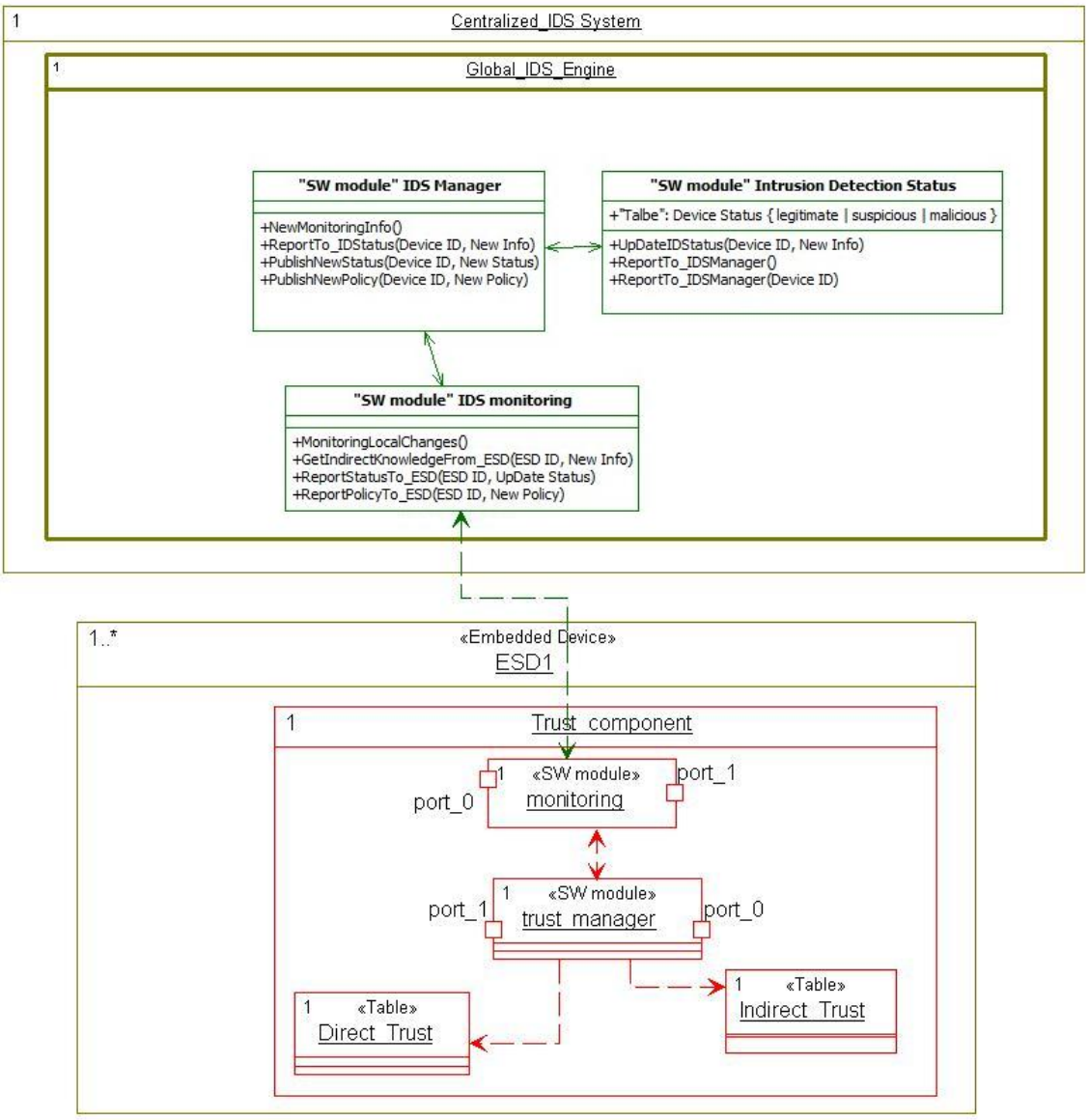

Figure 6-18: Global IDS (simplified deployment view)

In the centralized/global approach (Figure 6-18), the global IDS engine receives information from the trust component of the ESD and uses this knowledge to evaluate the status of each device registered in the network. If an intrusion is detected, the new status of the device(s) under question is broadcasted to the network and appropriate action is taken based on the predefined policy,

Anonymity and Location Privacy service

This section provides a suggested implementation of the anonymity service relying on the anonymizer server approach. It accommodates the location based services functionality together with the necessary SPD feature of anonymity for protecting system's identity and location privacy. Anonymity services might be accompanied by additional measures should they be considered necessary.

Note that although the *anonymizer* component is part of the nSHIELD network layer the computation overhead incurred might prohibit its deployment in any type of ESD. The suggested scheme should make

no assumptions about the network topology (only requiring a path from node to server via a distributed tree) and be as lightweight as possible given the required k-levels and monitoring accuracy

Dependencies:

The anonymizer might depend on the crypto component found on the node layer while the location privacy module depends on the anonymizer component. Additional measures might apply as previously mentioned.

Deployment View:

The client has the ability to request location based services via the corresponding adapter, depicted in Figure 6-19, either through an anonymizer, or directly, hence revealing to the server the client's true identity. The anonymizer component, which might be implemented as a discoverable service by a different ESD within the nSHIELD system, has to use the location data provided by the client, and will in turn format a request that will conceal/hide the client's true identity in an appropriate manner. The anonymizer might have to user similar location data from additional nodes in order to mix location data.

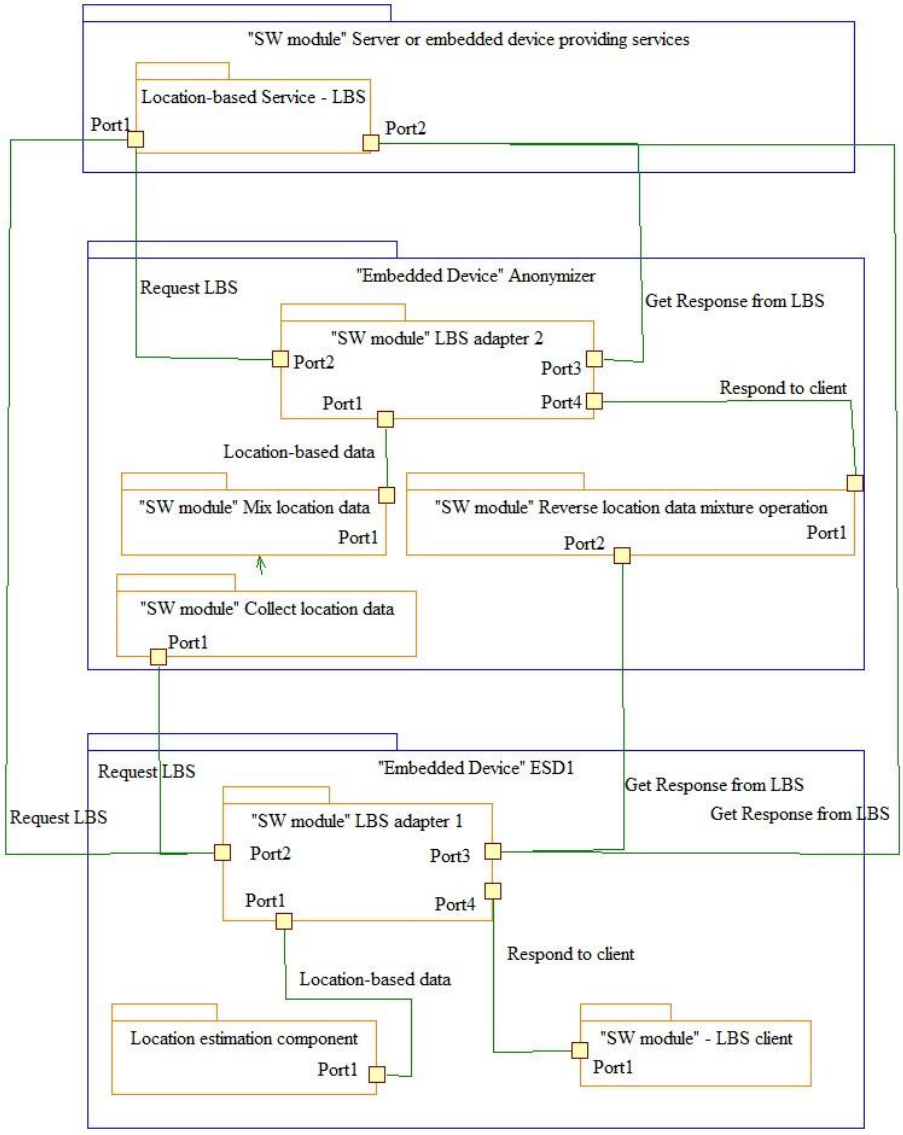

Figure 6-19: Anonymity and location privacy (simplified deployment view)

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.

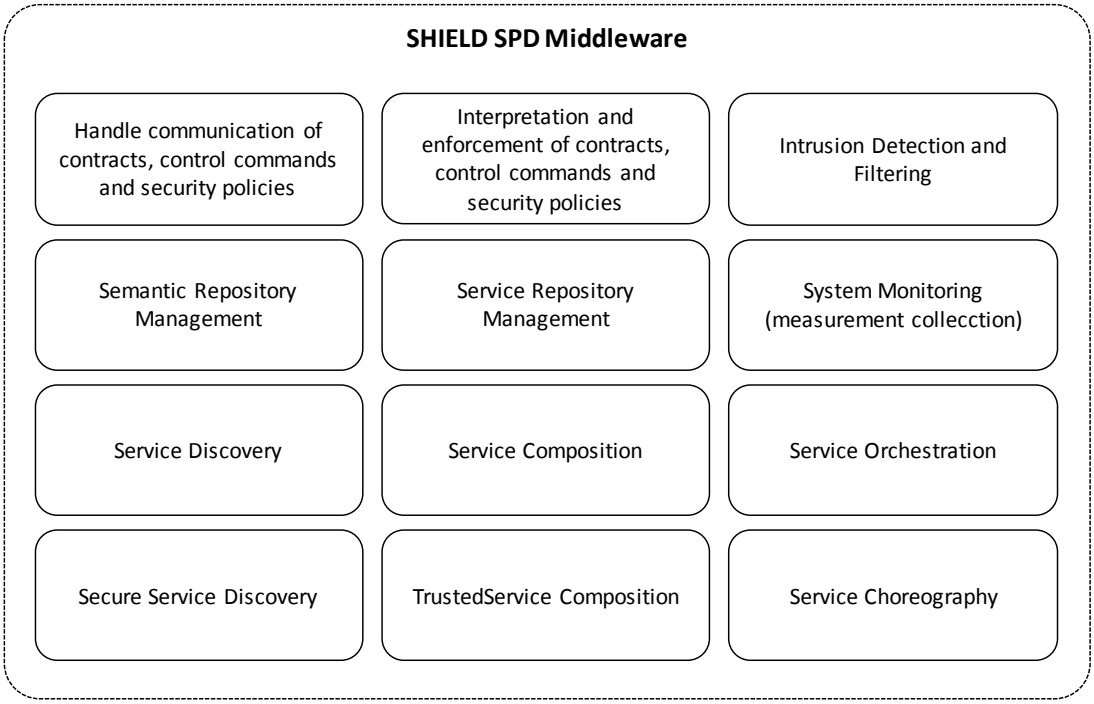

Figure 6-20: SHIELD Middleware services and functionalities

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 nSHIELD secure service discovery uses a variety of discovery protocols (such as SLP[5], SSDP[6], NDP[7], DNS[8], SDP[9], UDDI[10]) to 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 a 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 nSHIELD 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

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

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

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

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

[9] Bluetooth Service Discovery Protocol

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

works “on-line” and is continuously operating in background to monitor the SPD status of the running services.

6.4.1.4 Service choreography

Service choreography is a kind of trusted composition method that will enable intelligence to service composition. In nSHIELD we propose a SPD Metrics based service composition which determines the known domain. SPD metrics will guarantee the correct limits and boundaries of nSHIELD. Specially, it takes relevance from in all functional layers which metrics correspond to different measurements performed within them and composed by them.

The problem comes when there is an uncertainty in non-knows domains. Reputation based metrics should be developed for that. Some literature[11] has been devoted to uncertainty and reputation based technologies for trust factor generation. Choreography is based on the profitable composition between different elements which are not orchestrated by the human being but by the elements themselves. A learning path should be stated for auto-generating this choreography element.

In nSHIELD, the trust factor will be composed through metrics and the choreography is an intelligent module that will serve to compound SPD functional services from Overlay layer. This module will have a pre-set up nSHIELD path from which the systems will learn how to manage with unknown new sub-systems required collaborating with nSHIELD system. The choreography service is compliant to secure Service discovery (using protocols such as SLP, SSDP, NDP, DNS, SDP, UDDI). nS-SPD-ESD will be configured to be ready for choreography bundle. This is depicted in the following section. However, for legacy and unknown systems, the gateway (nS-ESD GW) will be the interface in order to require all the information for admitting an unknown and outsider element to be inserted in the choreographer bundle.

6.4.1.5 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 dominating 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.

6.4.1.6 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.

6.4.1.7 Handle communication of contracts, control commands and security policies

This functionality is the direct link with the SHIELD Overlay and is responsible of sending to 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.

6.4.1.8 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.

[11] Read Self x Technologies assessment in WP3

6.4.1.9 System Monitoring (measurement collection)

Together with services description and semantic metadata, also measurements could be provided to the Overlay. This task is in charge to a System Monitoring functionality that can inspect information flow related to internal (with regards to the middleware environment) and/or external interfaces. The System Monitoring is complementary to Service Discovery functionality since discovery is a request/response (asynchronous) protocol focussed to seek for specific, available services and/or capabilities, while system monitoring is a synchronous (periodic) task aiming at continuously collect and analyse incoming and outgoing data flows.

6.4.1.10 Intrusion detection ad filtering

This functionality is in charge of filtering all the discovery/composition requests/responses in order to detect potential menaces like DoS attacks. This functionality can be considered as complementary to the monitoring, and for this reason it could be implemented, on a deployment perspective, by the monitoring module itself.

6.4.2 Development and Deployment view

Basing on the nSHIELD components, that are applicable also for the nSHIELD Middleware, the SHIELD Middleware is located over the *SHIELD Adapter*.

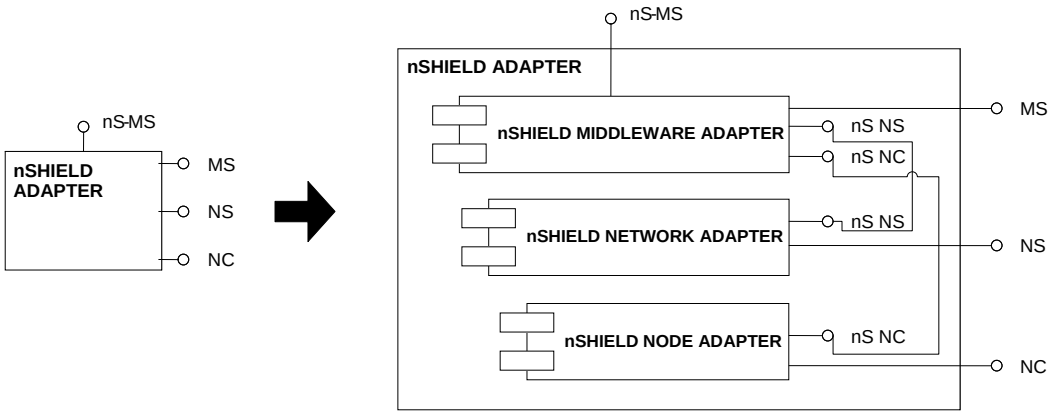

Figure 6-21: nSHIELD Adapter

The SHIELD Adapter is a technology dependent component in charge of interfacing with the legacy Node, Network and Middleware functionalities (through the MS, NS and NC interfaces). The legacy functionalities can be enhanced by the SHIELD Adapter in order to make them *SHIELD-compliant,* i.e. they become SDP *legacy device components*, which can be composed with other SPD components, according to the SPD Composability approach. In addition, the SHIELD Adapter includes *Innovative SPD functionalities* which are SPD *nSHIELD-specific components,* which can be composed with other SPD components. The SHIELD Adapter exposes the technology independent SHIELD Middleware layer functionalities that are used by the Security Agent component.

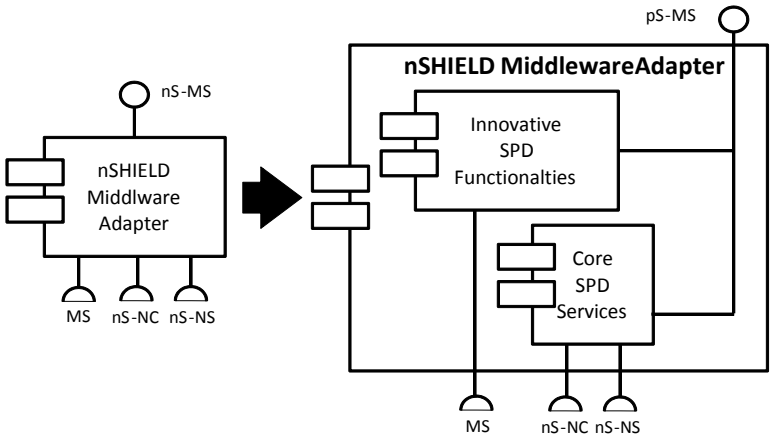

Figure 6-22: nSHIELD Middleware Adapter

As indicated in the logical view, the SHIELD Middleware Adapter can be further partitioned in the Core SPD services and in the Innovative SPD functionalities. These two components are linked through the nS-MS interface. The Core SPD Services are in charge to discover all the available functionalities at Node, Network and Middleware layers and to describe them in a technology-independent fashion. All the information is sent to the Overlay through the nS-MS interface. The nSHIELD Middleware Adapter should also carry into operation the decisions taken by the Overlay and communicated via the nS-MS interface by actually composing the discovered SPD functionalities. The nSHIELD Middleware Adapter includes a set of Innovative SPD functionalities interoperating with the legacy ESD middleware services (through the MS interface) in order to make them discoverable and composable SPD functionalities.

The deployment view that realises the Logical view described in the previous section is depicted in the following figure. Each module implements one or more functionalities.

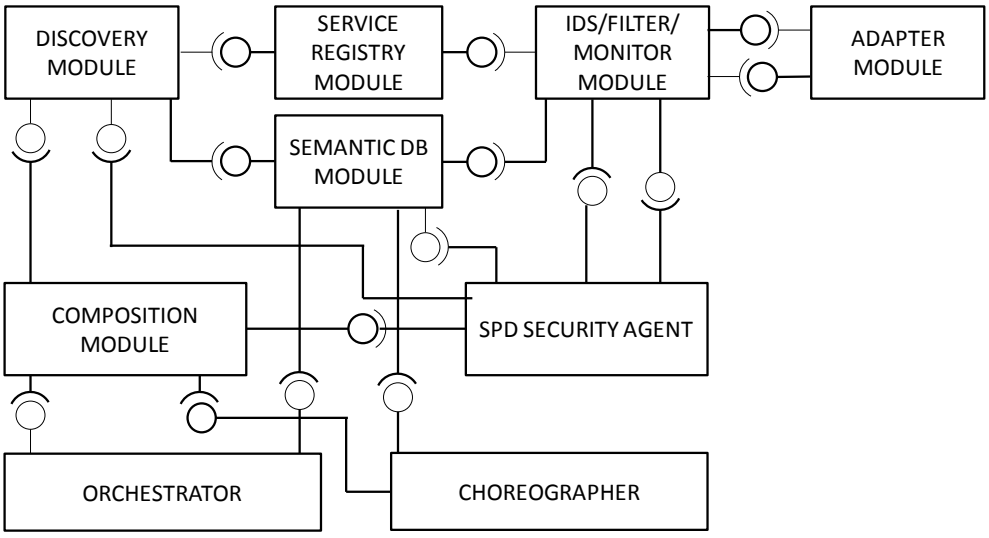

Figure 6-23 SHIELD Middleware: development view

In particular the mapping is the following:

Adapter Module:

- Handle communication of contracts, control commands and security policies
- Interpretation and enforcement of contracts, control commands and security policies

IDS/Filter/Monitor Module

- Intrusion Detection and Filtering
- System Monitoring (measurement collection)

Semantic DB Module

- Semantic Repository Management

Service Registry Module

- Service Repository Management

Discovery Module

- Service Discovery
- Secure Service Discovery

Composition Module

- Service Composition
- Trusted Service Composition

Orchestrator

- Service Orchestration

Choreographer

- Service Choreography

The *SPD Security Agent* will be addressed in the overlay section, while a description of the main components is provided below.

6.4.2.1 Adapter Module

The Adapter Module structure is depicted in the following figure:

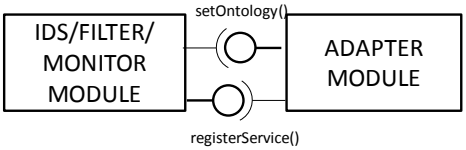

Figure 6-24 Adapter Module

- **Adapter Module**: it represents a generic (Node, Network or Middleware) nSHIELD Adapter for any type of legacy SPD functionality. The Adapter Bundle is in charge to:
    1. Provide an Innovative SPD functionality interacting with the underlying legacy services, capabilities and resources;
    2. register the provided Innovative SPD Functionality in the Service Registry using the *registerService()* interface;
    3. publish the semantic description of the Innovative SPD Functionality in the Semantic DB using the *setOntology()* interface;
    4. Manage information exchange to/from the other SHIELD components

Its messages are filtered by the IDS/FILTER/MONITOR for control and security purposes

6.4.2.2 IDS/Filter/Monitor Module

The IDS/Filter/Monitor Module structure is depicted in the following figure:

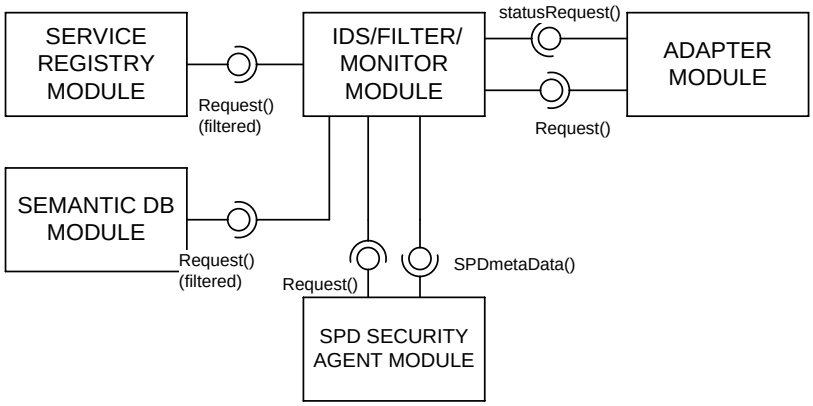

Figure 6-25 IDS/Filter/Monitor Module

- **IDS/Filter/Monitor Module:** it represents the collector of requests and measurements coming from the system (node, networks) to the middleware, and vice versa. Information are basically:
    1. requests for service registry and semantic DB module, managed by *request() (Filtered)* interfaces,
    2. Information for SPD composition to/from the security agent, via the *Request()* and *SPDmetaData()* interfaces;
    3. Measurements about the status of the SHIELD elements, for monitoring purposes (*statusRequest()* interface)

6.4.2.3 Semantic DB Module

The Semantic DB Module structure is depicted in the following figure:

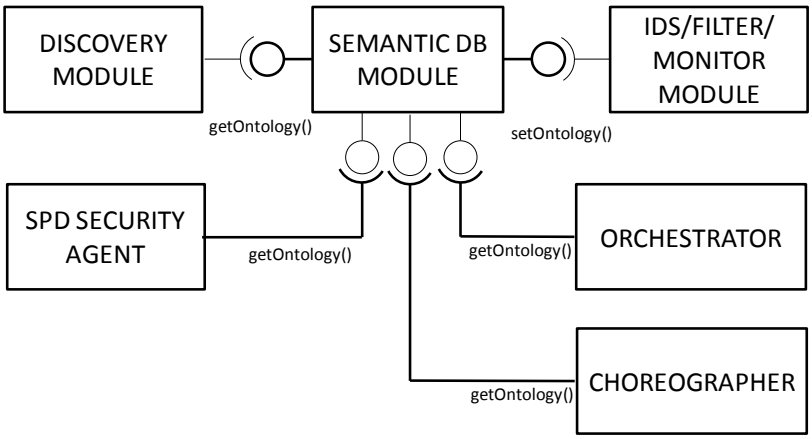

Figure 6-26 Semantic DB Module

- **Semantic DB Bundle**: it is in charge to store properly the semantic set by each Adapter Module (via the IDS/Filter/Monitor Module) through the *setOntology()* interface. The stored ontologies contains all the information to compose the available Innovative SPD functionalities. The Semantic DB Bundle provide access to the ontologies through the *getOntology()* interface.

6.4.2.4 Service Registry Module

The Service Registry Module structure is depicted in the following figure:

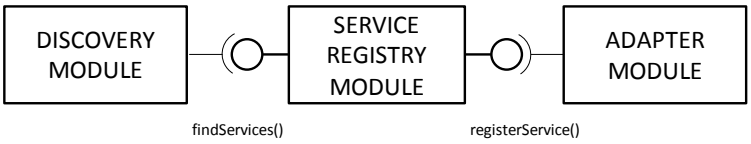

Figure 6-27 Service Registry Module

- **Service Registry Module:** it is in charge to store the bundle (i.e. SPD component) description in terms of provided functionalities, interfaces, semantic references, etc. Any SHIELD Node, Network or Middleware layer component can be registered here to be discovered by its own proper nSHIELD Adapter. The Adpater registers each bundle as a service, using the *registerService()* interface. The Service Registry provides the services entries information to the Discovery Bundle by means of the *findServices()* interface.

6.4.2.5 Discovery Module

The discovery module structure is depicted in the following figure:

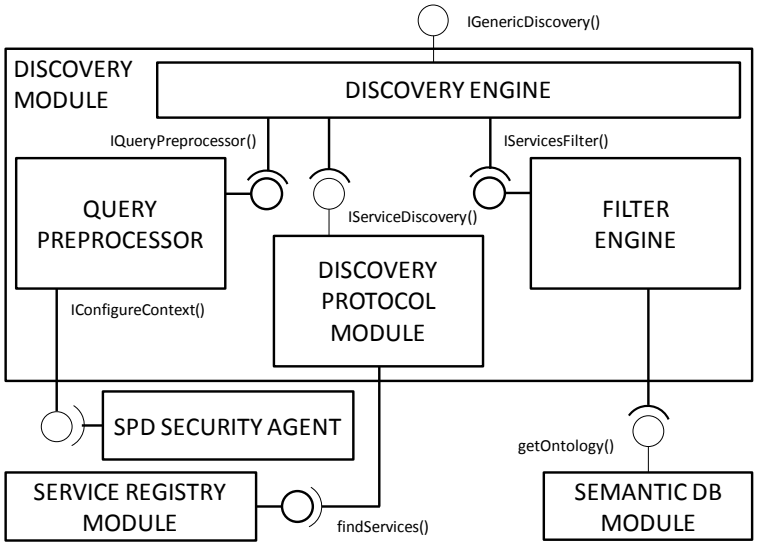

Figure 6-28 Discovery Module structure

As explained in the previous sections, the Discovery module is composed by the following components:

- **Discovery Engine**: it is in charge to handle the queries coming from the *IGenericDiscovery()* interface. The Discovery Engine manages the whole discovery process and activates the different functionalities of the Discovery service. It calls the *IQueryPreprocessor()* interface to enrich semantically and contextually the query. After that the query is sent to the different underlying discovery protocols, by means of the *IServiceDiscovery()* interface, to harvest over the interconnected systems all the available SPD components. Finally the list of discovered services is sent to the Filter Engine using the *IServicesFilter()* interface to discard those components not matching with the enriched query.
- **Query Preprocessor:** it is in charge to enrich the query sent by the Discovery Engine with semantic information related to the peculiar context. The query pre-processor can be configured

by the SPD Security Agent to take care of the current environmental situation using the *IConfigureContext()* interface;

- **Discovery Protocol:** it is in charge to securely discover all the available SPD components description stored in the Service Registry Bundle, using a the *findServices()* interface;
- **Filter Engine:** it is in charge to semantically match the query with the descriptions of the discovered SPD components. In order to perform the semantic filtering, the Filter Engine can retrieve from the Semantic DB the information associated to the SPD components, by means of the *getOntology()* interface.

6.4.2.6 Composition Module

The Composition Module structure is depicted in the following figure:

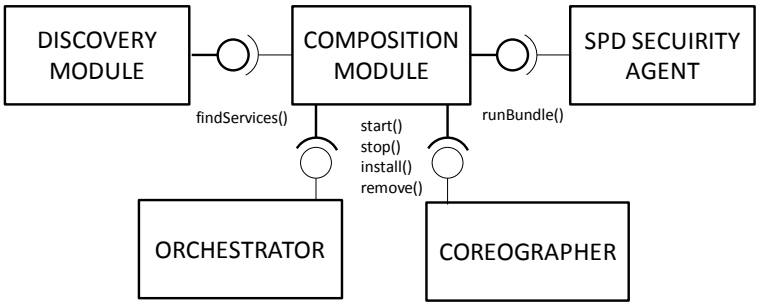

Figure 6-29 Composition Module

- **Composition Module**: it is in charge to compose the discovered bundles accordingly with the composition rules determined by the SPD Security Agent. Once the SPD Security Agent communicates through the *runBundle()* interface the necessity to run a composed functionality, the Composition Module use the *findServices()* interface to discover any suitable SPD component to be composed. Then the Composition Module compose the available bundles (taking care of the inter-bundle dependencies and the API-IMPL relationships) and uses the *start()*, *stop()*, *install()* and *remove()* interfaces provides by the Orchestrator and/or the choreographer.

6.4.2.7 Orchestrator/Choreographer

The development view of the orchestrator is the same as the choreographer in terms of interfaces, so they will be addressed together, as depicted in the following figure

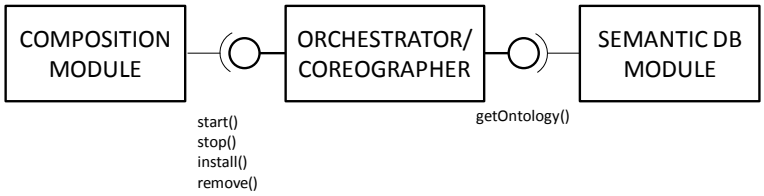

Figure 6-30 Orchestrator/Coreogrpaher Module structure

These modules share the same development view, since they are modules in charge of managing the composition of SPD functionalities according to the guidelines provided by the Security Agent in the semantic DB via the *getOntology()* interface. Service management is performed by means of *start()*, *stop()*, *install()*, *remove()* interfaces.

In a deployment perspective, this module can coexist, since it could be chosen, for example by policies or by design, to enable a centralized orchestration of services, or a distributed choreography. Further details on these mechanisms will be provided in WP5 deliverables.

6.5 Overlay

The Overlay is a logical vertical layer in charge of deciding, according to control algorithms and 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 nSHIELD Overlay offers five services, partially overlapped with the middleware ones. They are depicted in the following figure.

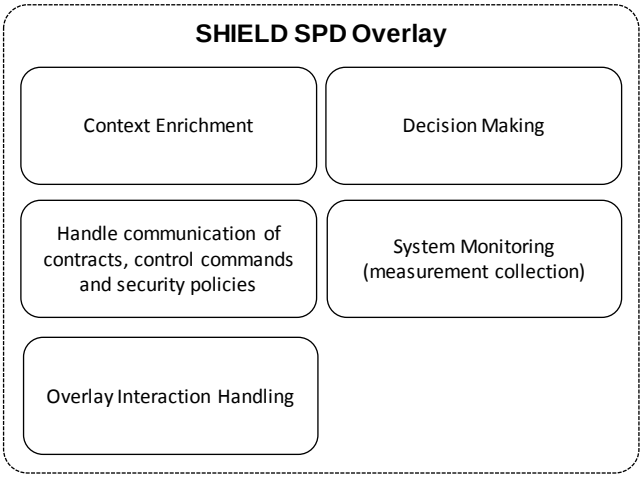

Figure 6-31: nSHIELD Middleware services and functionalities

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 6-32 provides a high level view of the internal structure of a security agent while it also depicts middleware layer components interacting with the overlay layer.

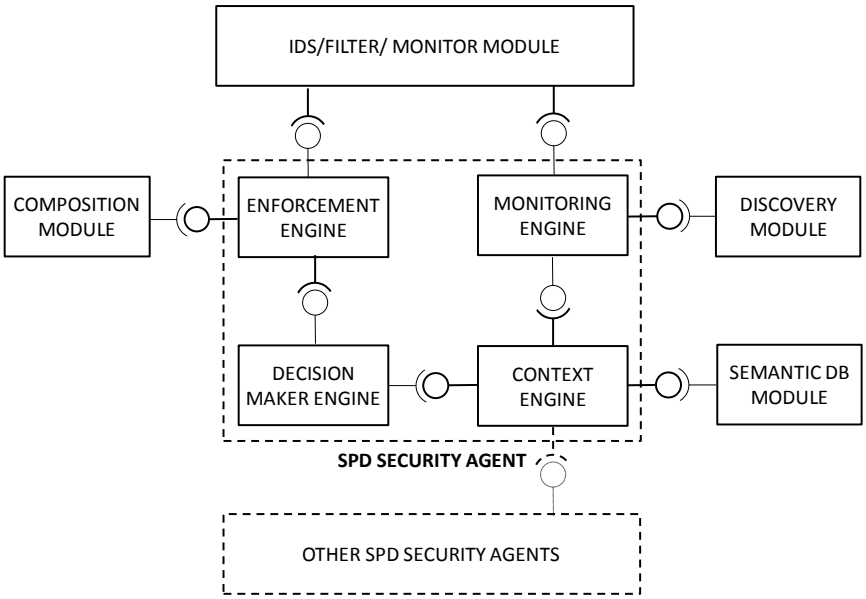

Figure 6-32: nSHIELD SPD Security agent architecture

The mapping between the functionalities described in the logical view and the component of the development view is the following:

Enforcement Engine

- Handle communication of contracts, control commands and security policies
  Monitoring Engine

- System monitoring
  Decision Maker Engine

- Decision making
  Context Engine

- Context enrichment
- Overlay Interaction Handling

A detailed description is provided below.

6.5.2.1 Monitoring Engine

The structure of the monitoring engine is depicted in the figure below.

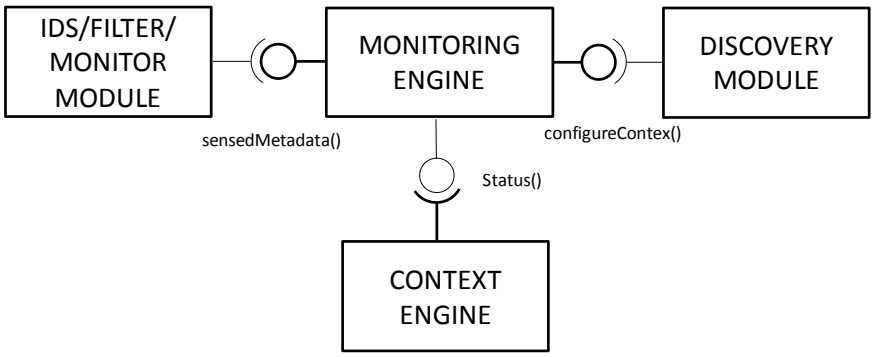

Figure 6-33 Structure of the monitoring engine

The heterogeneous data collected from the three horizontal layers (passing through the middleware layer via the IDS/Filter/Monitor Module) are abstracted and translated into technology-independent metadata. The resulting metadata (referred to as *sensedMetadata()*) are interpreted by the monitoring engine and stored in the context engine as *status()* of the system.

The monitoring engine is also able to drive the Discovery process by sending to the Discovery Module specific Context Information (*configureContext()* interface)

6.5.2.2 Context Engine

The structure of the context engine is depicted in the figure below.

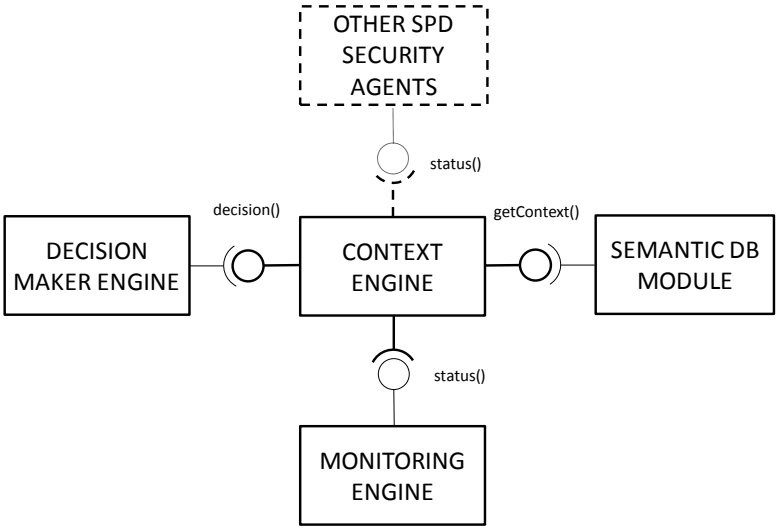

Figure 6-34 Structure of the context engine

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 context engine is also in charge of exchanging context information with the other security agents, in order to drive their decision in an autonomous, but aware way.

6.5.2.3 Decision Maker Engine

The structure of the decision maker engine is depicted in the figure below.

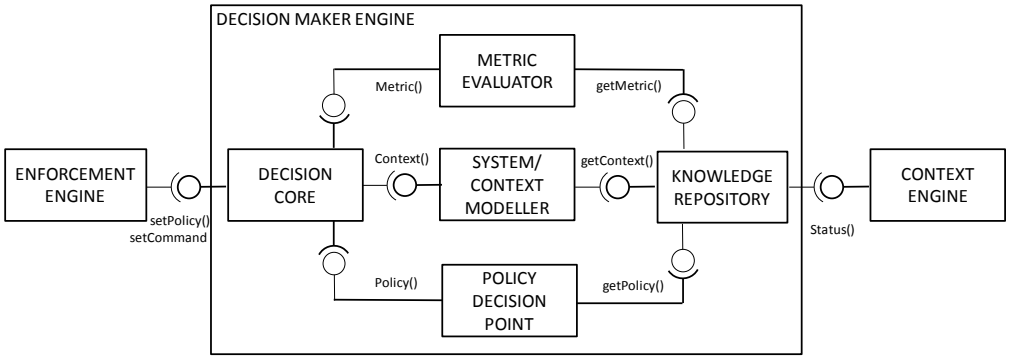

Figure 6-35 Structure of the decision maker engine

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.

In particular a modular approach is adopted for this component: the information retrieved by the context engine (including SPD metrics, SPD desired level and system information) are stored in a local knowledge repository, since each security agent should have access only to the information relevant to its controlled

cluster. Then, three modules running in parallel evaluate the desired composition (or composition constraints) of SPD functionalities with respect to:

- Metric (an algebric evaluation driven by metric approach)
- Policy (a set of predefined rules)
- System/context modeling (advanced control/optimization techniques)

Their outputs are send to a collector, the decision core, who (depending on the context and user needs) select the most appropriate composition of SPD functionalities. In a critical environment with no room for uncertainty, for example, the decision core will use mainly a policy-based approach, to have the situation under control in a pre-defined, deterministic way; in another scenario, where there are plenty of composition possibilities, and then a system/context structured approach may be used to perform additional optimization.

These mechanisms will be examined in detail in WP5 deliverables.

Finally, the commands are sent to the system components and the policies are sent to the policy enforcement points.

6.5.2.4 Enforcement Engine

The structure of the enforcement engine is depicted in the figure below.

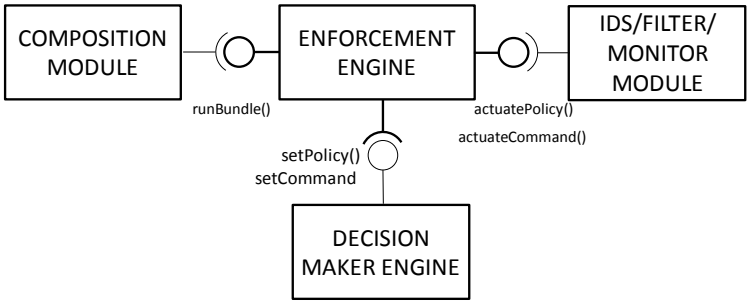

Figure 6-36 Structure of the enforcement engine

The decisions taken by the decision maker engine are sent to the enforcement engine (*setPolicy(), setCommand()*) and 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 (*runBundle()* interface to directly drive the composition, *actuatePolicy()* and *actuateCommand()* interfaces for filtered actions, or action needed to be translated by the adapters.).

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 7-1 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.) **(see 6.2.1.4.2)** **Secure Keys** from crypto key generator module to be used from secure data communication service (see 6.2.1.2.1)	**Measurements** (On request or periodically to compute SPD metrics) **(see 6.2.1.4.2)**	–
Network layer (→)	**Commands** To configure/ reconfigure transmission related parameters of the RF channel (smart transmission service) **(see 6.2.1.3.1)**	–	**Measurements** (On request or periodically to compute SPD metrics) **(see 6.4.2.2 & 6.4.2.1)**	–

Middleware layer (→)	**Commands** (for composition or configuration of SPD modules) Measurement Requests (to get system status) **(see 6.2.1.4 & 6.4.2.2 & 6.4.2.1)**	**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) Measurements Requests (to get system status) **(see 6.4.2.2 & 6.4.2.1)**	–	**Sensed (SPD) metadata** (to feed control algorithms and decision making modules) **(see 6.5.2.1)**
Overlay layer (→)	–	–	**Rules for composition and discovery (see 6.5.2.4)**	–

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	–	–
Middleware layer (→)	–	–	Commands to manage middleware services	Status of the node SPD capabilities, network and middleware SPD services
Overlay layer (→)	–	–	Requests to actuate Policies and commands **(see 6.5.2.4)**	Aggregated context information **(see 6.5.2.2)**

Table 7-2 consists a first approach in defining the type of information exchanged between different embedded nodes with respect to the 4 nSHIELD functional layers. The information currently provided is quite high level and is based on the deployment diagrams provided in section 0. It should be reviewed, refined and detailed in future versions of the architecture deliverable based on feedback from implementation WPs (WP3, WP4, WP5).

7.3 Components

The present document does not provide detailed information regarding the flows and the precise API calls between the various components within any of the nSHIELD functional layers. This was not possible since the deliverable provides the reference architecture without delving deeply in the internal implementation of

the various services. The process view is not yet available while the deployment view provides, in most cases, a high level description of the components aiming mainly to serve as a reference for a more detailed analysis that should take place within WP3, WP4, WP5. Still, the various deployment diagrams of chapter 0, reveal in some cases information regarding the interfaces and data that may be exchanged between the various components within a specific layer. More information is expected to become available as activities in WP3, WP4 and WP5 evolve and feedback is provided in WP2. This will enable a more detailed listing of component interfaces that will possible be included in the last version of the architecture deliverable (D2.7).

As an example, for Figure 6-8 (example deployment view at node layer) the following interfaces and data flows between components could be defined:

- Status information flows from all components to the status component in the monitor module.
- Key information flows from cryptographic component to the anti-tamper component (actually this happens at a logical view, as at implementation view it passes through the MCU)

In the case of the network layer the various deployment diagrams provide some information on both the type of messages and the information that needs to be exchanged while the same stands for the middleware and overlay case.

8 Application Scenarios Realization

The 4 nSHIELD scenarios (Railroad Security, Voice/Facial Verification, Dependable Avionic System, Social Mobility and Networking) were used, so far, 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 was the 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 system architecture for each application scenario. Such use cases may contain the following data:

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

As a first step, in the section below we provide a possible realization of the nSHIELD system reference architecture to one of the scenarios defined in the Technical Annex, namely the one that relates to railway Security Management System (SMS). Through it we seek to demonstrate the compatibility of the reference nSHIELD architecture with the overall SMS concept. The section provides also additional details on the actual SPD services and capabilities that are needed for implementing the SMS use case. These capabilities evidently are (should be) a subset of the ones defined throughout sections 6.2 to 6.5.

8.1 Railway Security Scenario

The basic concept and needs of the Railway Security application scenario have been adequately described in nSHIELD D2.1 deliverable [3]. The foreseen system, a smart-surveillance system suitable for the protection of urban or regional railways, includes a multitude of devices and subsystems that provide capabilities like: (a) intrusion detection and access control, (b) intelligent video-surveillance, (c) intelligent sound detection and (d) integrated management of all resources from a central location. The scenario also prescribes the use of an appropriate network infrastructure that will enable secure communication and interconnection of all distributed smart-sensors installed along the railway line both in fixed (e.g. bridges, tunnels, stations, etc.) and mobile (passenger trains, freight cars, etc.) locations with a security control center.

Figure 8-1 provides an overview of the assets that need protection and the overall environment context of this Security Management System (SMS). Technically, the collaborative grid is composed by near field communication devices that collect and transmit information through larger scale networks (i.e. WANs, GSM or GSM-R) to administrative and security control points. The monitoring devices, consisted of sensors and cameras, have as a main function the transmission of alarms in the event of any abnormal situation, as the latter is defined by the system's policies.

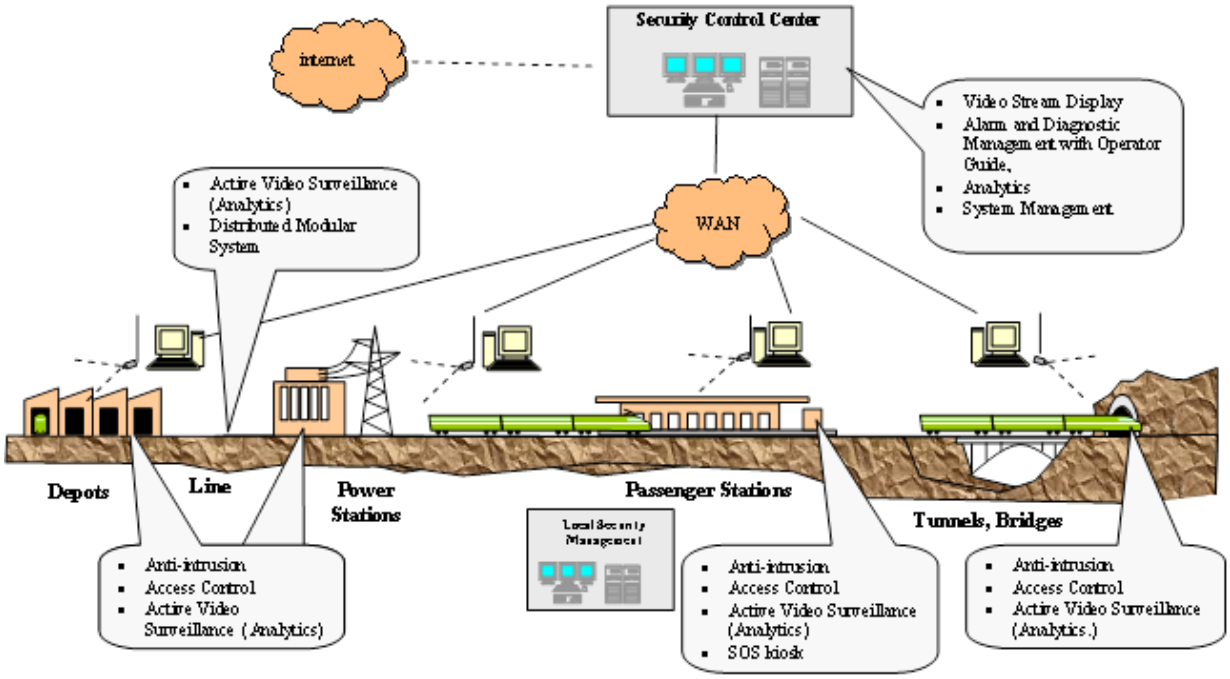

Figure 8-1: The overall SMS concept

8.1.1 Compatibility to nSHIELD architecture

In order to realize the SMS on the basis of the nSHIELD reference architecture that was presented in section 0 and prove its compatibility with it, Figure 8-1 must be transformed so that it provides a more network oriented viewpoint of the overall system. This is done in Figure 8-2 which focuses more on the network topology and a distinction of the various subsystems involved. All sub-systems are connected through dedicated and completely segregated network links. Real-time communication between the on-board and the ground is allowed by a wide-band wireless network.

Compatibility of the SMS architecture as depicted in Figure 8-2 with the nSHIELD overall architecture (Figure 6-4) is evident. Considering the Railway Security Management System from a network topology point of view we can consider that SMS adopts also a hierarchical approach where all devices under a station LANs map to nSHIELD sub-system while a train LAN can be realized either as an nSHIELD aware cluster that connects and reports to a station LAN (i.e. while a train at a station pier) or even as a subsystem by itself (i.e. while train en-route).

The sensing devices of SMS are mostly categorized as *L-ESD*s although many of them (especially the ones at station LAN level) can be converted into smart-sensors by integrating the sensor unit with the nSHIELD processing units (both hardware and software). More powerful nodes, on the top of depot level (SMS Clients, CBRNE and Access Control servers etc.), mainly fall under the category of *nS-ESD* although at least one of them should also host a *Security Agent* software allowing thus the interconnection of all station and train LANs to the “nSHIELD Overlay” cloud but also monitoring and enforcement of policies from the command & control centre.

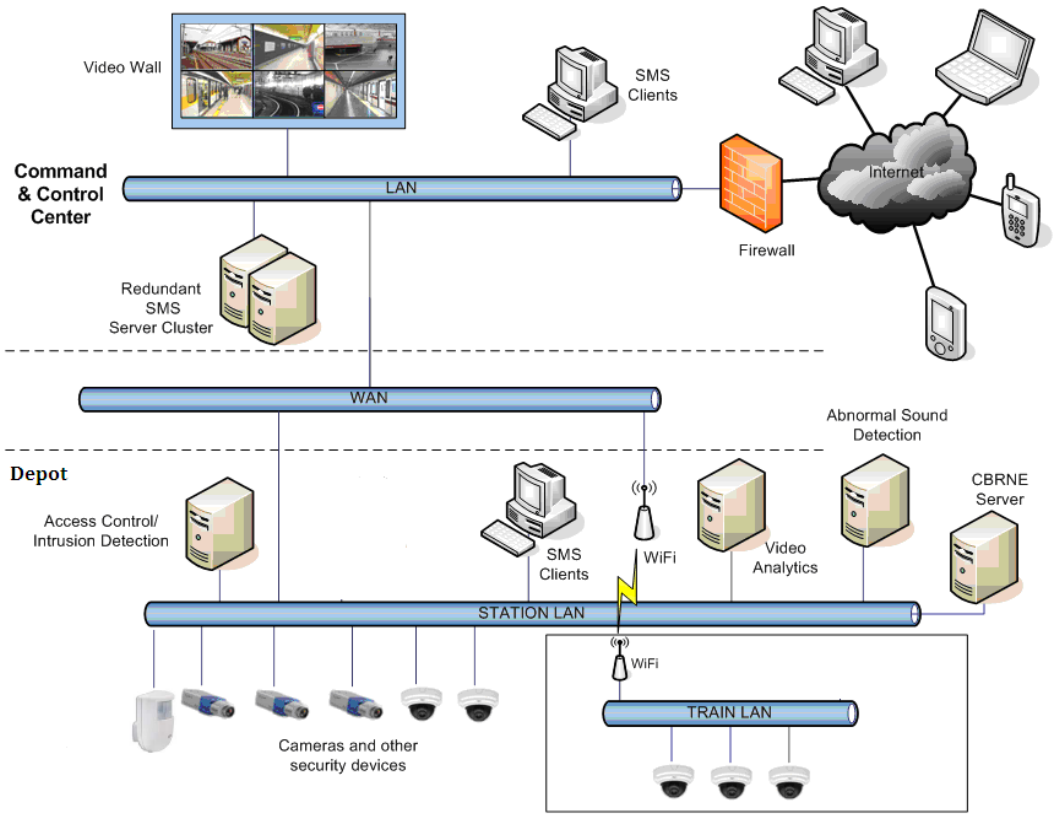

Figure 8-2: SMS Architecture

8.1.2 Required SPD Services and Functionalities

Rail-based mass transit systems are vulnerable to many criminal acts, ranging from vandalism to terrorism. Therefore, physical security systems for infrastructure protection comprises all railway assets as for tunnel, train on board, platform and public areas, external Areas, technical control room, depots, electrical substations and etc…

A rough categorization of the services provided by nSHIELD Architecture in the realization of Railway Security scenario yields (without excluding situations of identicalness or functions belonging to both categories):

1. The objectives are to forecast critical threats as: aggressions and abnormal behaviours, sabotage and terrorism, vandalism and graffiti, thefts and pick-pocketing
2. Services related to the establishment of the needed SPD level infrastructure: ensuring that data flow is routed through a secure and dependable network

These services are what we call, in the context of the project, “SPD functionalities” and are further categorized per layer responsible for granting them. For the Railway SMS application based on a risk analysis that has been performed and with respect to the analysis of SPD capabilities in sections 6.2 through 6.5 an indicative list of supported capabilities include:

- Power management: the nodes used for monitoring functionalities must have a power supply in case of blackout. In order to guarantee the reliability of monitoring system.
- Status monitoring: It is responsible to monitor nodes of system and alert in case of dependability or security fault.
- TPM: is useful to guarantee a node as trusted in its HW and SW capabilities. In order to forecast security attack.

- Access Control: with the use of authentication protocol is possible to attest the introduction of malicious node or devices in the network (digital signature). Avoid the acces of unauthorized peopole in order to forecast tamper attack to system devices.
- Cryptography: good service to avoid unsecure communication between node or node/server.
- Trusted connectivity (including*Trusted Routing, secure data exchange)*:
- Intrusion Detection support (at network level)
- Service provision and composition: from discovery to orchestration

In other words, the targeted outcome of the use of nSHIELD System instances is to ensure *security* and *trustworthiness*, both in the physical path that people, materials and infrastructure are moving or operating and in the virtual path that information about these functions is communicated.

8.1.3 Example nSHIELD Application

Currently, the security system described above is highly heterogeneous, sensors differ in their inner hardware-software architecture and thus in the capacity of providing information security and dependability. Natural and malicious faults can impact on system availability and indirectly on safety, since the SMS is adopted in critical infrastructure surveillance applications.

The problems mentioned above can be solved by adopting the nSHIELD architecture. In more details this includes:

1. **Security attack:** a malicious node is inserted into the monitoring system, in order to capture or send wrong/invalid information. Currently, SMS doesn't provide a mechanism of authentication at node level. The use of nSHIELD solution can be useful for this situation.
2. **Secure communication:** the trusted connectivity, access control and use of cryptography guarantee a high level of security.
3. **Fault of node:** currently in SMS is possible to detect the fault of component and reconfigure the system. With the service *Status monitoring* (SM) is possible to detect the status of node by metrics. In case of node fault, before of its MTTF, the SM has different choices:
    a. to stop the system in order to substitute o repair the node;
    b. to reconfigure the system in presence of redundant nodes. It can calculate the MTTF of other nodes, if values are feasible it can reconfigure the monitoring functionalities with the remain nodes.

9 Conclusions

Based on the preliminary architectural framework, yielded in D2.3, this deliverable (D2.4) has proceeded in the definition of nSHIELD System Reference Architecture. Some of the components and modules were completely reworked and others were slightly modified, whereas more analytical views were added.

The cornerstone of the overall architecture proposal has remained the definition of its 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*.

In accordance with the rest of 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 a formalized reference system architecture and sets the technical challenges, the open issues and subsequent work plan, in the way to finalize nSHIELD System Architecture. The major improvements from D2.3 contents are: the addition of *Development and Deployment View* for all layers, the drastic revision of Node layer, further definition of information flows and interfaces and a first attempt to visualize an application scenario (Railway Security) using the services of nSHIELD architecture. Continuing in T2.3 the proposed structure will be under continuous improvement, through the interaction with the theoretical framework of requirements and metrics (WP2), the technical developments that will support the architecture (WP3, WP4 and WP5) and the validation and test procedures (WP6). The already once updated list of Interfaces shall be further refined 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 complete the link of the theoretical architectural framework with real implementations, providing thus its proof of concept. In this context, nSHIELD system 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] D2.5 “Preliminary SPD Metrics Specifications” , nSHIELD project deliverable, September 2012
[5] OSGi Alliance. Osgi service platform specification overview. http://www.osgi.org/resources/spec overview.asp, 2003.
[6] 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.
[7] Sun Microsystems. Jini network technology. http://wwws.sun.com/software/jini/, 2003.
[8] W3C. Web services architecture. http://www.w3.org/TR/2003/WD-ws-arch-20030808/, 2003
[9] Bertrand Meyer. Contracts for components. Software Development, 2000.
[10] Antoine Beugnard. Jean-Marc jezequel et al. Making Components Contract Aware, IEEE Journal Computer archive Volume 32 Issue 7, July 1999
[11] Tammy Noergaard, Embedded Systems Architecture, A Comprehensive Guide for Engineers and Programmers, Elsevier Inc. 2005
[12] http://www.middleware.org/whatis.html
[13] The official CORBA standard from the OMG group – http://www.omg.org/spec/CORBA/Current/
[14] DCOM Remote Protocol Specification – http://msdn.microsoft.com/library/cc201989.aspx
[15] 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
[16] The Hydra project – http://www.hydramiddleware.eu
[17] Prism-MW - Architectural Middleware for Mobile and Embedded System – http://csse.usc.edu/~softarch/Prism/
[18] http://www.embeddedlibrary.com/Embedded_Science/Embedded_Systems/Embedded_Systems_Design_and_Development_Lifecycle.html
[19] IEEE 1471 "Recommended Practice for Architectural Description of Software-Intensive Systems"
[20] IEEE ISO/IEC 42010:2007, Systems and Software Engineering---Recommended practice for architectural description of software-intensive systems
[21] Ministry of Defense Architecture Framework (MoDAF), http://www.mod.uk/DefenceInternet/AboutDefence/CorporatePublications/InformationManagement/MODAF/ModafMetaModel.htm
[22] Department of Defense Architecture Framework (DoDAF), http://dodcio.defense.gov/dodaf20.aspx
[23] The Open Group Architecture Framework (TOGAF), http://www.togaf.info/
[24] ISO Reference Model for Open Distributed Processing (RM-ODP), http://www.rm-odp.net/
[25] 4+1 View Model, http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf
[26] 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.
[27] 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
[28] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. New Jersey: Prentice Hall, 1992.
[29] V. T. Raisinghani and S. Iyer, “Cross-Layer Design Optimizations in Wireless Protocol Stacks,” *Computer Communications*, vol. 27, pp. 720–724, 2004.
[30] A. S. Tanenbaum, *Computer Networks*, 3rd ed. Prentice-Hall, Inc., 1996.
[31] L. Larzon, U. Bodin, and O. Schelen, “Hints and Notifications,” in *Proc. IEEE Wireless Communications and Networking Conference (WCNC’02)*, Orlando, 2002.
[32] 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.
[33] 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.

[34] 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.
[35] 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
[36] Vijay T. Raisinghani, Sridhar Iyer: “Cross Layer Feedback Architecture for Mobile Device Protocol Stacks”,
[37] Vineet Srivastava and Mehul Motani Srivastava: “Cross-Layer Design: A Survey and the Road Ahead”, IEEE Communications Magazine, Dec 2005.
[38] Giovanni Giambene and Sastri Kota: “Cross-layer protocol optimization for satellite communications networks: A survey”, Int. J. Satell. Commun. Network. 2006
[39] 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
[40] R. Winter et al., “CrossTalk: A Data Dissemination-Based Crosslayer Architecture or Mobile Ad Hoc Networks,”
[41] V. T. Raisinghani and S. Iyer: “ECLAIR: An efficient cross layer architecture for wireless protocol stacks”, WWC2004,
[42] Yana Bi, Mei Song, Junde Song: “Seamless mobility Using Mobile IPv6”, Publication Year: 2005
[43] 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
[44] 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
[45] Srivastava, V., Motani, M.: “The Road Ahead for Cross-Layer Design”, Publication Year: 2005 , Page(s): 551 – 560.
[46] IETF: Policy Framework [Online], http://datatracker.ietf.org/wg/policy/charter/
[47] Verma D.C.: “Simplifying network administration using policy-based management”, IEEE Network, 2002, pp. 20-26
[48] ETSI DES 282 001 V0.0.1 (2006-09)
[49] 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
[50] T. Charles Clancy, Nathan Goergen, “Security in Cognitive Radio Networks: Threats and Mitigation”
[51] 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
[52] Jan Peleska, “Formal Methods and the Development of Dependable Systems”\
[53] Martin ARNDT, “Towards a pan European architecture for cooperative systems”, ETSI Status on Standardization, 2009
[54] AbdelNasir Alshamsi, Takamichi Saito, “A Technical Comparison of IPSec and SSL”, Tokyo University of Technology
[55] Your Electronics Open Source, “Embedded Systems in SDR and Cognitive Radio”, http://dev.emcelettronica.com/
[56] 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
[57] G. Wiederhold, Mediators in the Architecture of Future Information Systems, In IEEE Computer XXV(3), pp. 38-49, 1992
[58] Kirk Martinez, Jane K. Hart, and Royan Ong. Environmental sensor networks. Computer, 37(8):50–56, 2004
[59] 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
[60] V. Casola, A. De Benedictis, A. Mazzeo and N. Mazzocca, SeNsIM-SEC: security in heterogeneous sensor networks, May 2011, SARSSI2011
[61] TinyOS Project, URL: http://www.tinyos.net
[62] 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

[63] Damasio A. (2000), “The feeling of what happens - body, emotion and the rise of consciousness”, Harvest Books
[64] 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
[65] 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
[66] 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
[67] 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
[68] 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
[69] 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
[70] Sensor model language (SensorML). (2005)[online], http://www.opengeospatial.org/standards/sensorml
[71] B. Adida, M. Birbeck (2008). RDFa primer: Bridging the human and data webs. [online] available: http://www.w3.org/TR/xhtml-rfda-primer
[72] R. Khare (2006), Microformats: The next (small) thing on the semantic web? Internet Computing 10(1), 68-75
[73] 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
[74] S. Marti, T. J. Giuli, K. Lai, and M. Baker, “Mitigating routing misbehavior in mobile ad hoc networks,” in Proceedings of the 6th annual international conference on Mobile Computing and Networking (MobiCom ’00), 2000, pp. 255–265.
[75] Krontiris, I., Dimitriou, T., Freiling, F.C.: Towards intrusion detection in wireless sensor networks. In: Proceedings of the 13th European Wireless Conference, Paris, France (April 2007).
[76] R. Roman, J. Zhou, and J. Lopez. Applying intrusion detection systems to wireless sensor networks. In Proceedings of IEEE Consumer Communications and Networking Conference (CCNC '06), pages 640--644, Las Vegas, USA, January 2006.
[77] Khoshgozaran, A., Shahabi, C., Shirani-Mehr, H.: Location Privacy: Going beyond k-Anonymity, Cloaking and Anonymizers. Journal of Knowledge and Information Systems 26(3), 435–465 (2011)
[78] http://technet.microsoft.com/en-us/library/jj131725.aspx “Trusted Platform Module Technology Overview”
[79] Molina-Jiménez C. and Marshall L.: True Anonymity without Mixes. In *Proceedings of the Second IEEE Workshop on Internet Applications (wiapp '01)* (WIAPP '01). IEEE Computer Society, Washington, DC, USA, 32-,(2001)
[80] Chaum D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM, 4(2), February (1981).
[81] Zillman M. P.: Privacy Resources and Sites on the Internet, (2012) online: http://whitepapers.virtualprivatelibrary.net/Privacy%20Resources.pdf
[82] Balfanz D., Durfee G., Shankar N., Smetters D., Staddon J., and Wong H.: Secret handshakes from pairing-based key agreements. In 24th IEEE Symposium on Security and Privacy, Oakland, CA, May, (2003)
[83] Castelluccia C., Jarecki S., and Tsudik G.. Secret handshakes from ca-oblivious encryption. In Advances in Cryptology - ASIACRYPT 2004, volume 3329 of Lecture Notes in Computer Science, pages 293-307. Springer, 2004.
[84] Jarecki S., Kim J., and Tsudik G.. Authentication for Paranoids: Multi-Party Secret Handshakes. In ACNS'06, June 2006.

[85] S. Jarecki, J. Kim and G. Tsudik, "Group Secret Handshakes Or Affiliation-Hiding Authenticated Group Key Agreement", CT-RSA}, pp. 287-308, 2007.
[86] S. Jarecki and X. Li, "Private Mutual Authentication and Conditional Oblivious Transfer". In: Halevi, S. (ed.) Advances in Cryptology - CRYPTO 2009. LNCS, vol. 5677, pp. 90-107. Springer, Heidelberg (2009)
[87] M. Abadi, "Private authentication". In Proceedings of the Workshop on Privacy Enhancing Technologies (PET 2002), San Francisco, CA, April 2002.
[88] M. Abadi and C. Fournet, "Private Authentication", Theoretical Computer Science 322, 3 (September 2004), 427-476.
[89] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In Proceedings of the 1st international conference on Mobile systems, applications and services, MobiSys '03, pages 31–42, New York, NY, USA, 2003. ACM.
[90] Software Communication Architecture (SCA) specification http://jpeojtrs.mil/sca/Documents/SCAv4_0/SCA_4.0_20120228_ScaSpecification.pdf