Project no: 269317

nSHIELD

new embedded Systems arcHItecturE for multi-Layer Dependable solutions

Instrument type: Collaborative Project, JTI-CP-ARTEMIS

Priority name: Embedded Systems

D5.1: SPD middleware and overlay technologies assessment

Due date of deliverable: M6 – 2012.02.29

Actual submission date: M10 – 2012.06.30

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

Organisation name of lead contractor for this deliverable:

Selex Elsag, SE

Revision [Issue 1]

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2012)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	X

Document Authors and Approvals

Authors		Date	Signature
Name	**Company**		
Andrea Fiaschetti	UNIROMA1		
Alberto Isidori	UNIROMA1		
Gaetano Scarano	UNIROMA1		
Roberto Cusani	UNIROMA1		
Salvatore Monaco	UNIROMA1		
Francesca Cuomo	UNIROMA1		
Antonio Pietrabissa	UNIROMA1		
Martina Panfili	UNIROMA1		
Andrea Morgagni	SE		
Renato Baldelli	SE		
Harry Manifavas	TUC		
Alexandros Papanikolaou	TUC		
Konstantinos Fysarakis	TUC		
Georgios Hatzivasilis	TUC		
Dimitrios Geneiatakis	TUC		
Konstantinos Rantos	TUC		
Inaki Eguia	TECNALIA		
Balázs Berkes	S-LAB		
Zoltán Hornák	S-LAB		
Mónika Halmy	S-LAB		
Roberto Uribeetxeberria	MGEP		
Dimitrios Serpanos	ATHENA		
Nikos Priggouris	HAI		
Ignasi Barri Vilardell	ISL		
Ljiljana Mijic	THYIA		
Nastja Kuzmin	THYIA		
Francesco Cennamo	SG		
Mariana Esposito	ASTS		
Reviewed by			
Name	**Company**		
Josef Noll	MAS		
Approved by			
Name	**Company**		
Elisabetta Campaiola	SE		

Applicable Documents

ID	Document	Description
AD1	TA	nSHIELD Technical Annex
AD2	D5.2	pSHIELD Semantic Technologies Report
AD3	D5.4	pSHIELD Middleware and Overlay Report
AD4	D5.1	pSHIELD Semantic Technologies Prototypes
AD5	D5.3	pSHIELD Middleware and Overlay Prototypes
AD6	D6.1	pSHIELD Platform development report

Modification History

Issue	Date	Description
Draft A	29.2.2012	First version
Draft B	26.3.2012	Draft A and review
Draft C	04.05.2012	Contributions to Chapter 4 added
Draft D	15.05.2012	Contributions to Chapter 4 content and: TUC and S-LAB
Issue 1	30.06.2012	Contributions to Chapter 5, 6. Final Issue

Contents

Figures

Tables

Glossary

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

1 Executive Summary

The SHIELD Roadmap is an R&D initiative funded under the ARTEMIS-JU programme, whose objective is to address security issues in the Embedded Systems domain, with particular focus on development, certification and composability of heterogeneous SPD technologies.

Due to the budget allocation, this roadmap has been divided in two phases: the pSHIELD[1] pilot project and the nSHIELD[2] full project. The first phase, closed on 31st December 2011, has produced significant achievements in terms of:

- Theoretical studies ([AD2][AD3])
- Prototypes ([AD4][AD5])
- Demonstration ([AD6])

These achievements represent a proof of concept that justifies and triggers the complementary activities that has to be carried out in the phase two, whose (long term) objectives are: i) the consolidation of the SHIELD guidelines, ii) the finalization of the SHIELD framework and iii) the demonstration in several, industrially relevant, application scenarios.

The purpose of the current document, as earliest (WP5 T0+1) outcome of WP5 activities, is to assess the results obtained by the pSHIELD investigation with respect to Middleware technologies, and to identify the potential research/improvement that will be investigated during the nSHIELD prosecution, in order to fully cover the SHIELD roadmap needs.

The main motivation of this deliverable, as indicated also in the technical annex (see box below), is to clearly define the boundaries between the two projects, while assuring an exhaustive approach.

[…] At the start-up of the project an assessment will be done on the technologies that the pSHIELD project has examined, in order to full address the right technologies for the nSHIELD project .[…]

In particular, D5.1 takes in inputs the scenario/user needs (that justify the development of the SHIELD framework), the outcome of phase one (as baseline) and the Technical Annex (for future development). This rationale is depicted in Figure 1.1.

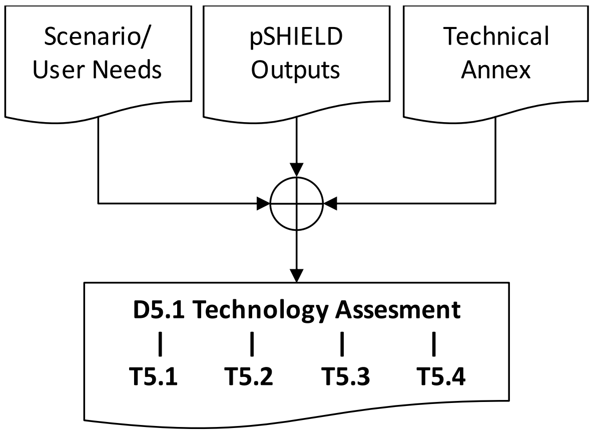


Figure 1.1: D5.1 rationale

[1] Call ARTEMIS-2009-1

[2] Call ARTEMIS-2010-1

Since this document will be the guideline for the prosecution of the work, it will be structured as follows: after an introduction, four sections, each one covering one task, will be considered; then each section will be further divided in two halves: <what has been achieved> and <what will be taken into account>, with one paragraph per technology.

Finally, each technology description will be associated to the corresponding text from the technical annex, to assure 100% coverage of Middleware activities.

Some preliminary links with the scenario needs will be identified as well (even if their definition is still in progress), in order to show the motivation behind specific technological choices.

2 Introduction

In this section, the rationale behind the SHIELD Middleware technologies, as defined in previous section, is reported, in terms of:

i. scenario/user needs,
ii. WP5 statement of work (technical annex)
iii. pSHIELD outcomes.

This will justify the work that had to be carried out.

2.1 Scenario/user needs

Nowadays, complex systems are integrated with a great engineering effort, trying to harmonize heterogeneous technologies and tailor them on the specific solution; in this context, security, privacy or dependability issues are faced only *a posteriori*, time by time, without a structured or standardized approach. This results in lack of reconfigurability or reusability of technical solutions and, above all, in the impossibility of assessing, *a priori*, the SPD level for a given system.

The scenario/user need behind this research is to provide industrial actors with the SHIELD platform, a framework that:

- Will offer ***innovative SPD functionalities***.
- Will enable the ***dynamic, scalable, modular, reconfigurable*** *and* ***measurable composability*** of SPD functionalities in the Embedded Systems domain

The activities carried out in WP5 will be finalized to create a Middleware able to satisfy these two topics.

Some example taken from the application scenario identified for the nSHIELD project will help to clarify better the importance of these needs.

2.1.1 SPD composability in railways surveillance

The first application scenario comes from the railways surveillance domain. 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…

The objectives of a surveillance system are to forecast critical threats as: aggressions and abnormal behaviours, sabotage and terrorism, vandalism and graffiti, thefts and pickpocketing.

A modern smart-surveillance system suitable for the protection of urban or regional railways is made up by distributed smart-sensors and several subsystems performing different functionalities:

1. Intrusion detection and access control:
   - volumetric sensors for motion detection;
   - magnetic contacts to detect illicit doors opening;
   - glass break detectors;
   - microphonic cables for fence/grill vibration detection;
   - active infrared barriers for detecting intrusions inside the tunnels;
2. Intelligent video-surveillance and Intelligent sound detection:
   - advanced cameras with special features;
   - digital video processing and recording, using efficient data compression protocols;
   - video-analytics of the scenes, using computer vision algorithms;
   - Microphones.

3. Dedicated communication network

4. Integrated management system

Distributed smart-sensors are installed along the railway line both in fixed (e.g. bridges, tunnels, stations, etc.) and mobile (passenger trains, freight cars, etc.) locations. They are integrated locally using local wired/wireless infrastructures (see Figure 2.1).

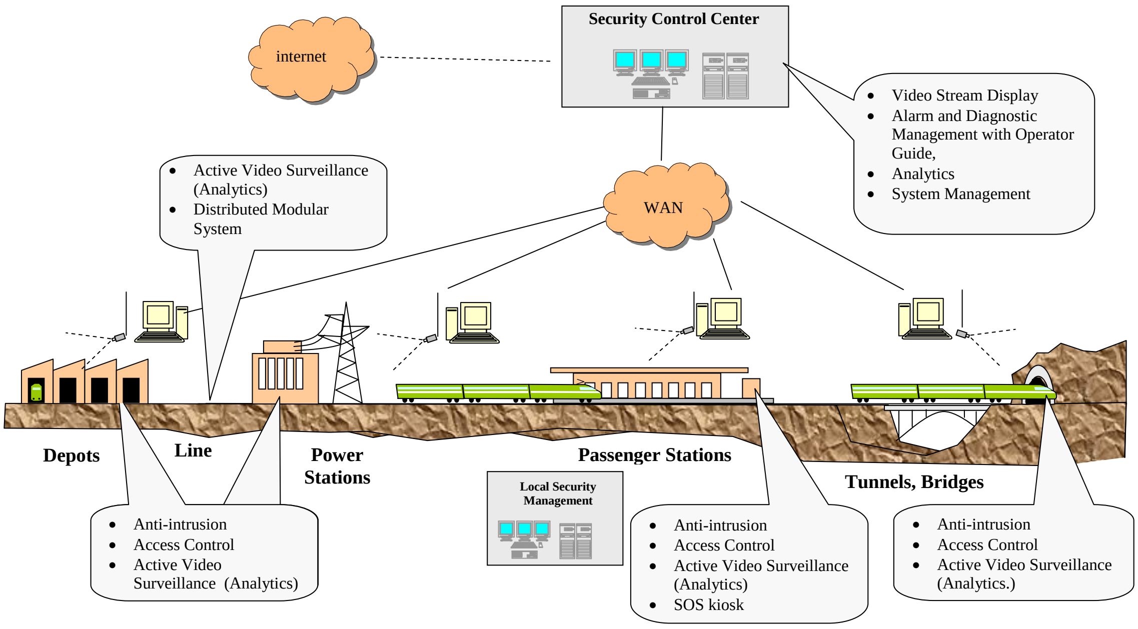

internet
Security Control Center
Video Stream Display
Alarm and Diagnostic Management with Operator Guide,
Analytics
System Management
Active Video Surveillance (Analytics)
Distributed Modular System
WAN
Depots
Line
Power Stations
Passenger Stations
Tunnels, Bridges
Local Security Management
Anti-intrusion
Access Control
Active Video Surveillance (Analytics)
Anti-intrusion
Access Control
Active Video Surveillance (Analytics)
SOS kiosk
Anti-intrusion
Access Control
Active Video Surveillance (Analytics.)

Figure 2.1: Typical monitoring architecture

Currently, the security system described above is highly heterogeneous in terms not only of detection technologies (which will remain such) but also of embedded computing power and communication facilities. In other words, the sensors that are put together differ in their inner hardware-software architecture and thus in the capacity of providing information security and dependability. This causes several problems:

- Information security must be provided according to different mechanisms and on some links - which are not "open" but still vulnerable to attacks - information is not protected by cryptographic nor vitality-checking protocols;
- Whenever any new sensor needs to be integrated into the system, a new protocol and/or driver must be developed and there is no possibility of directly evaluating the impact of such integration on the overall system dependability;
- New dedicated and completely segregated network links often need to be employed in order not to make the sensor network exposed to information related threats;
- The holistic assurance and evaluation of dependability parameters (e.g. for assessment/certification purposes) would be a very difficult task.

In particular both natural and malicious faults can impact on system availability and indirectly on safety, since the surveillance system is adopted in critical infrastructure surveillance applications.

Manage Heterogeneous Technologies | Perform Composition | Evaluate security impact of a component | Assess overall security

The problems mentioned above can be solved by adopting the SHIELD architecture for the surveillance framework, i.e. architecture able to compose, in a seamless way, heterogeneous technologies and to quantify the individual security impact, as well as to assess the overall security strength of the system. This will be mainly in charge of Middleware technologies.

2.1.2 SPD composability in avionic surveillance

Another SHIELD application is the surveillance and monitoring performed by UAVs (see Figure 2.2).

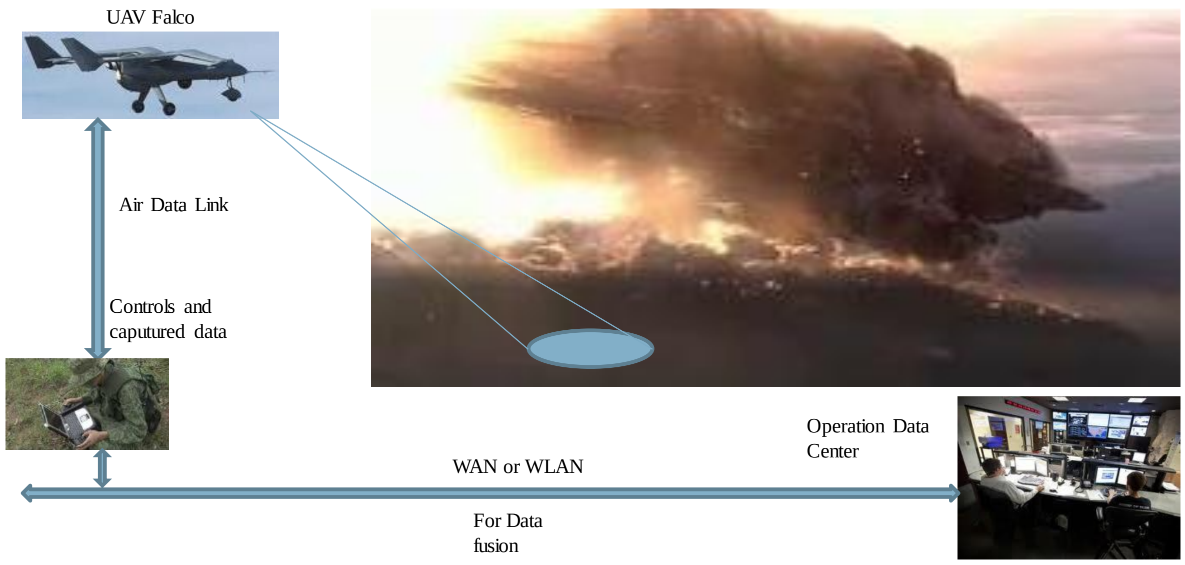


Figure 2.2: UAV surveillance

They are usually applied for several purposes:

Remote sensing: Biological sensors are sensors capable of detecting the airborne presence of various microorganisms and other biological factors. Chemical sensors use laser spectroscopy to analyse the concentrations of each element in the air

Commercial aerial surveillance: Aerial surveillance of large areas is made possible with low cost UAV systems. Surveillance applications include: livestock monitoring, wildfire mapping, pipeline security, home security, road patrol and anti-piracy.

Oil, gas and mineral exploration and production: UAVs can be used to perform geophysical surveys, in particular geomagnetic surveys where the processed measurements of the differential Earth's magnetic field strength are used to calculate the nature of the underlying magnetic rock structure. Knowledge of the underlying rock structure helps trained geophysicists to predict the location of mineral deposits. The production side of oil and gas exploration and production entails the monitoring of the integrity of oil and gas pipelines and related installations.

Scientific research: Unmanned aircraft are uniquely capable of penetrating areas which may be too dangerous for piloted craft. An UAV can fly into a hurricane and communicate near-real-time data directly to the National Hurricane Center.

Search and rescue: Search for missing person. A concept of coherent change detection in SAR images allows for exceptional search and rescue ability: photos taken before and after the storm hits are compared and a computer highlights areas of damage.

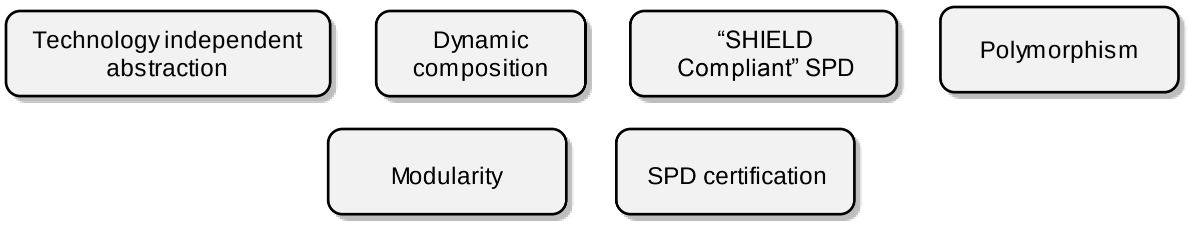
Since these systems are quite recent in their application, they suffer from several problems, including:

- Heterogeneous networks are not fully supported, so there is the need to improve information exchange capabilities

- Partial integration of devices, only with same family systems
- Complex Certification
- Redundancy is obtained only by duplicating HW
- Trusted Data communication is possible only with old protocols and algorithms
- Scalability is not foreseen, so every time the system must be improved, added engineering effort is needed
- Expensive integration of different standards and different vendors
- Sensitive to cyber attacks

The problems mentioned above can be solved by adopting the SHIELD approach. In particular communication between non homogeneous data and integration of different devices will be enabled by the technology independent abstraction; redundancy will be improved by enabling the dynamic composition of both hardware and software technologies that are “SHIELD Compliant”; resilience to cyber-attacks will be achieved by enabling system polymorphism (UAVs architecture changes continuously); scalability will be enabled by the modularity of the SHIELD framework; the certification process will be eased by the adoption of recognized standard (like the Common Criteria) as baseline for SPD evaluation.

All these issues will be mainly in charge of Middleware technologies.



2.1.3 Remarks

These two examples are explicit enough to outline the problems that affect complex systems engineering, in the embedded systems domain and for security relevant applications. It must be said that Middleware technologies will be focused on the creation of the “enabling capabilities” that make the SHIELD framework work. The SPD improvements (i.e. the development of new SPD functionalities) will be mainly in charge to WP3 and WP4, since these tasks are more technology oriented.

In the prosecution of the document the highlighted boxes will be mapped over the Middleware solutions, to justify the technological choices.

2.2 WP5 Statement of Work

The second input for D5.1 is the Technical Annex, i.e. the Statement of Work for WP5 Activities, from which the following text is fully taken.

Objectives

The objectives of WP5 are:

- *Define a common semantic to describe the SPD interfaces and functionalities;*
- *Improve SPD middleware technologies;*
- *Provide support to legacy SPD systems;*
- *Introduce the Overlay concepts and functionalities;*
- *Develop a prototype to be integrated in the demonstrators.*

Task 5.1: SPD driven Semantics

In this task semantic technologies will be developed to address the interoperability among different SPD technologies. A semantic ontology will be defined to describe the SPD components and functionalities, to describe the metrics and the exchanged information between the node, network, middleware and overlay layer.
The designed semantic language may be used also to represent profiles and policies according to interoperable and self-describing formats. The exploitation of semantic technologies will allow meaningfully representing and reasoning about context and policy information.
The outcome of this task will be a lightweight common semantic languages derived by standard ones (OWL) in order to be easily processed in the embedded system world where the processing unit are limited in power and resources.
The semantic ontology will be part of the prototypes delivered by WP5.

Task 5.2: Core SPD services & Adaptation of Legacy Systems

This task will design and/or develop the core SPD services provided by the SHIELD middleware:

- *service discovery entailing mechanisms to securely register, advertise, discover, locate, filter, rank and select the available services;*
- *service composition entailing mechanisms to automatically resolve the dependencies and to discover, deliver and deploy the atomic services;*
- *service orchestration entailing run-time mechanisms to install, start, pause, stop, refresh and uninstall the services in a distributed environment.*
- *context awareness features to refine and extend the existing middleware orchestration functionalities in order to improve their performance (strict liaisons with Task 5.4);*

The interaction between SPD-middleware and ES nodes will be bidirectional. The SPD-middleware will use data received by ES nodes and will provide information to the upper layers of the system. In both cases information passing through the middleware (e.g. information, configurations and data) should be represented in a proper way (e.g. /by using semantics) in order to enable features for providing advanced service discovery, composition and orchestration functionalities to the applications.
A complete architecture will be defined for developing the SHIELD core SPD services, which will be based on the exchange of semantic metadata, used to describe, for each service, its model, the relevant security requirements and the needed SPD components.
A prototype of the proposed architecture will be implemented, a key feature of which will be the exploitation of ontology-based, semantic technologies to represent the SPD services model. Since an integrated support to SPD functionalities in middleware architectures is still largely unexplored in both academic and industrial research activities, this task will investigate how to extend the firstly emerging models to accomplish with the SHIELD requirements provided by WP2. Indeed the design and development of the SPD Middleware core SPD services will be accomplished according to the specifications, requirements and architectural guidelines coming from tasks 2.1 and 2.3.
In order to build the target functionalities a modular approach will be followed by partition the features of the middleware in abstract SPD modules. Each module will be a collection of conceptually similar functions that provide services to other modules or layers: the SPD modules will be characterized to be dynamically composable.
This task will also define and implement specific interfaces (based on the design results of task 2.3) for accessing middleware capabilities from outside the system. The SPD modules will be implemented as software modules which will become part of the prototypes delivered by WP5 on M18 and M30.

This task foresees also the design of highly-dependable interfaces and/or, adapters and/or enablers to make heterogeneous legacy SPD solutions (protocol, standards, mechanisms, techniques, etc.) for ES nodes, networks and middleware interwork transparently with the enhanced capabilities provided by the SHIELD approach. The main outcome of this task will be the design of adapters to make legacy devices capable to support the SHIELD SPD-functionalities, as well as the development of some prototypal software.

Task 5.3: Policy-based management

This task aims at designing and developing a SPD-middleware policy-based management for ensuring a high level of security, privacy and dependability in systems composed by Intelligent ES Nodes (developed in WP3) and based on Smart Transmissions (developed in WP4) on the base of the metrics identified in task 2.2. In order to build specific management functionalities and procedures for accomplishing these objectives, several aspects will be investigated and analysed. The main ones are:

- *Use of policies. Policies permit the declarative specification of security strategies separately from the implementation code of ES nodes. The use of interpreted policies allows to change the security behaviour of a node without recoding or shutting down the node;*
- *Design and development of algorithms and tools to enrich the smart capabilities of the middleware and increase its autonomy;*

The outcome of task 5.3 will be integrated in the WP5 prototypes.

Task 5.4: Overlay monitoring and reacting system by security agents

This task aims to design and implement an overlay layer based on a system of reacting security agents. The outcome of this task will be a software implementation of a security agent prototype ready to be integrated and interwork with the rest of SHIELD architecture.
The security agent will be designed to interpret and elaborate the SPD information generated by the SHIELD multi-layer framework. So the Security Agent produces high-level SPD information which are aggregated and eventually shared and distributed with other Security Agents acting with different scopes to the SHIELD systems. The high-level SPD information will be assessed with the metrics defined in task 2.2, in order to assess the SPD level of the single layer as well as of the overall system.
The security agent will be designed and developed to build autonomously an overlay network composed by different security agents that monitor SPD among groups of embedded system peers, networks, applications an d services. Each security agent will interpret the information shared in the SPD system in order to discover imminent threats, menaces and vulnerabilities. All security events of interest will be correlated with the underlying criticality rating the targeted asset. This will results in accurate prioritization and enables fast response to the threats, targeting most critical assets.
The security agent reacting system will be a combination of network scanning, passive network monitoring, and integration with existing data provided by the layers. It allows the security agent to organize the network assets into categories. This feature will permit to assign ad-hoc security policies for monitoring each application or service component.
A multi-agent approach which combines intelligent, adaptive, autonomous and cooperative capabilities of the agents will be developed. Teams of security agents will cooperate to monitor over time the SPD level on the whole service chain. Therefore, in order to guarantee security and dependability in inter-agent communication, new semantically enriched communication protocols and distributed algorithms capable of dynamically identifying potential dangerous activities, will be analyzed and defined.
The benefits brought by semantic technologies developed in Task 5.1 will be also adopted to exploit the security agent capability and adapt security needs and associated policies to possible unforeseen situations.
The main outcome of this task will be the development of a software prototype (on M18 and M30) ready to be integrated in the SHIELD platform.

2.3 pSHIELD outcomes

The pSHIELD pilot phase has already produced several studies and prototypes of functionalities and components at Middleware level. The outcomes, in terms of technologies examined within the WP5, are depicted in Figure 2.3.

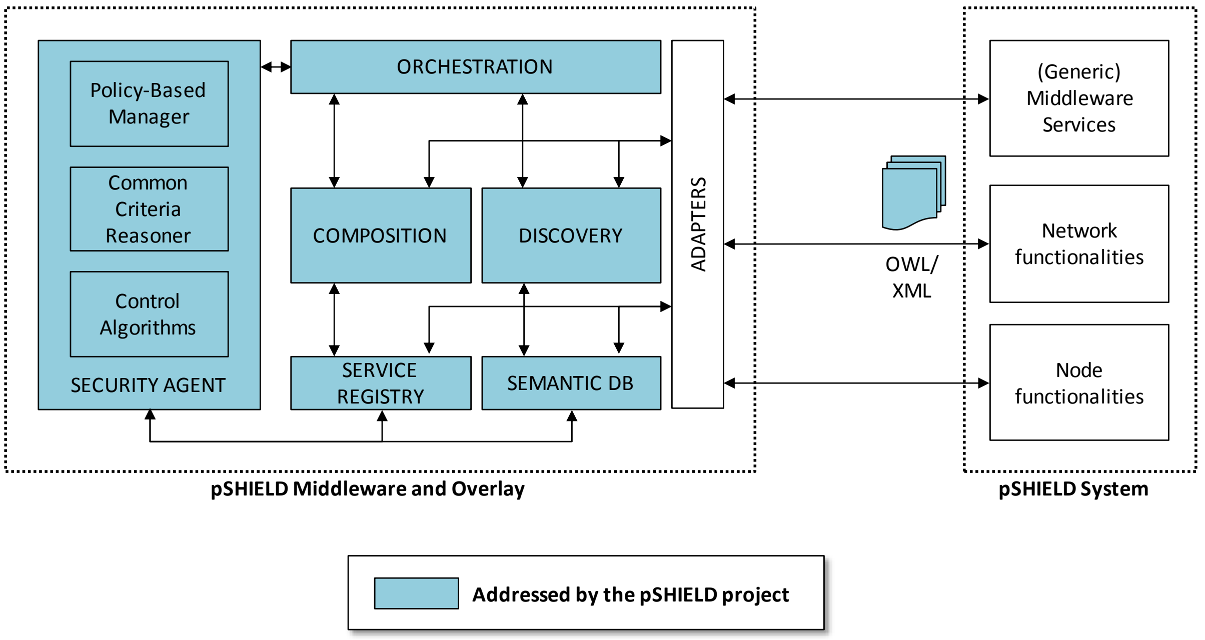


Figure 2.3: pSHIELD outcomes

In particular they are:

- A methodology to derive the SHIELD semantic and a preliminary ontology in OWL language, tailored on a significant but reduced scenario (railways domain)
- A framework that implements middleware core services for discovery, registration, orchestration and composition of SHIELD components, developed in Java language (based on OSGI)
- An engine that performs SPD composability based on a Common Criteria-like metric
- Architectural design and performances analysis of a Policy Based approach by which the middleware composition could be driven
- A theoretical foundation and Matlab simulations of an Hybrid Automata-based approach to drive the SPD composition in a context-aware way
- The definition, at architectural level, of the behaviour of the Security Agent

It is evident that some work must be still done to fully achieve the Technical Annex objectives mentioned in the previous section, in all potential scenarios, so these outcomes are only the starting point of the prosecution of the research. nSHIELD will go further in two ways, by means of:

- **Improvements**: some technologies will be enriched (new or further implementation)
- **Innovativeness**: new technologies will be developed

The differences between the pilot phase and the full project are depicted in Figure 2.4, where the outcomes expected from nSHIELD are highlighted.

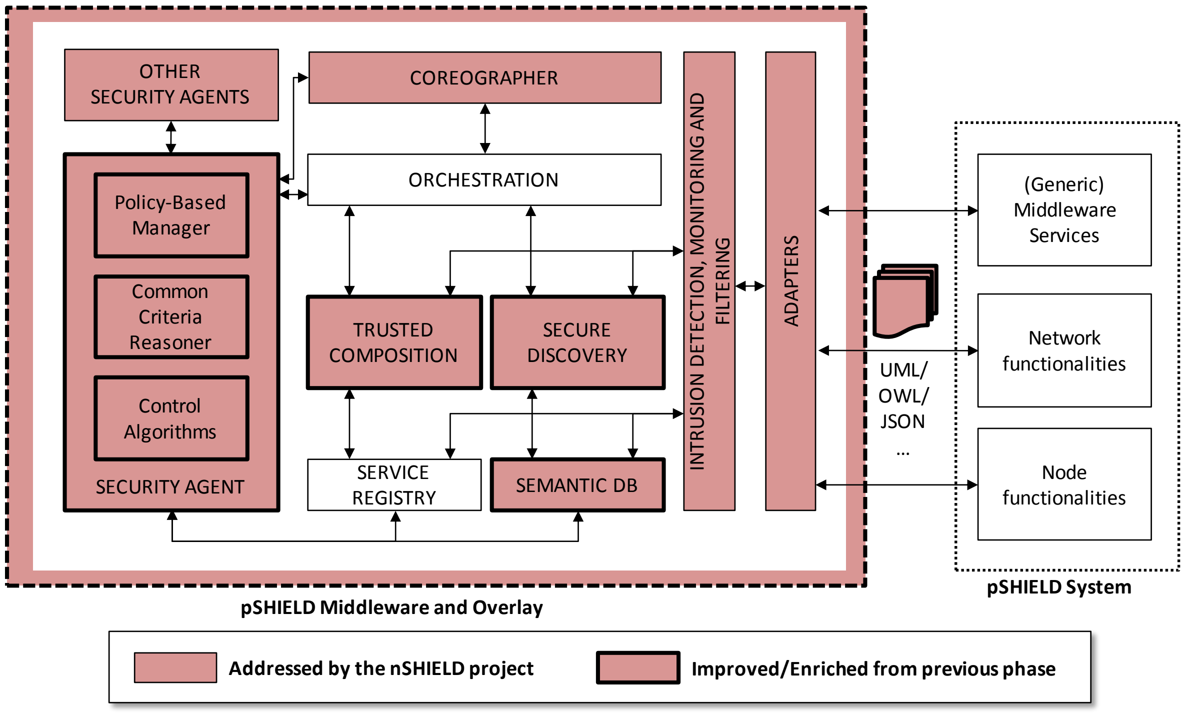


Figure 2.4: nSHIELD outcomes

In particular they are:

- The enrichment of the semantic approach by means of new languages and new procedures to represent and manage the information necessary to enable the composability
- The improvement of the SHIELD middleware core services, with the enabling of a "secure" discovery and a "trusted" composition.
- The identification of new middleware core services, like the "choreographer" and the "intrusion detection monitor and filter".
- The definition of the adapters, by which the SHIELD system is able to interface the external world (and in particular legacy devices)
- The enrichment of the Security Agent architecture, with the harmonization of all the approaches (standardized, policy-based or context aware) that drive the composability
- The definition of the interactions between several Security Agents working together to manage SPD in distributed environments
- The development of innovative control algorithms, based on DES and Petri Nets, to model and control the system behaviour
- The instantiation of the Policy Based management architecture, as well as the definition of several libraries of policies to manage SPD in different application scenarios.
- The certification of the protection profile for the SHIELD middleware (Common Criteria compliant).

These technologies, as well as their heritage from the pilot project, will be analysed in the prosecution of the document.

3 Semantic technologies assessment

3.1 pSHIELD results and adopted technologies

In the pilot phase the major technological achievements in semantic technologies were:

- The formalization of a procedure to define the pSHIELD model
- The design of a SHIELD ontology in OWL language

3.1.1 Procedure to define the pSHIELD semantic model

The challenge of the SHIELD roadmap is to model the SPD context by means of semantic technologies. This representation should be generic enough to fit as much scenarios as possible, but at the same time to provide sufficient details for the assessment of the security aspects that generally are strictly linked to the specific application.

To achieve these objectives, the approach derived in the pilot phase is based on three guidelines:

1. the *translation* of the *real word* into a uniform technological description.
2. the *representation* of *functional properties* by means of ontology as well
3. the *identification* of the *relations* between real/structural and functional world.

So, as depicted in Figure 3.1, the problem of modelling SPD in the context of ES is reduced to the formulation of three different meta-models describing: i) structure, ii) functions, iii) relations between structure and functions.

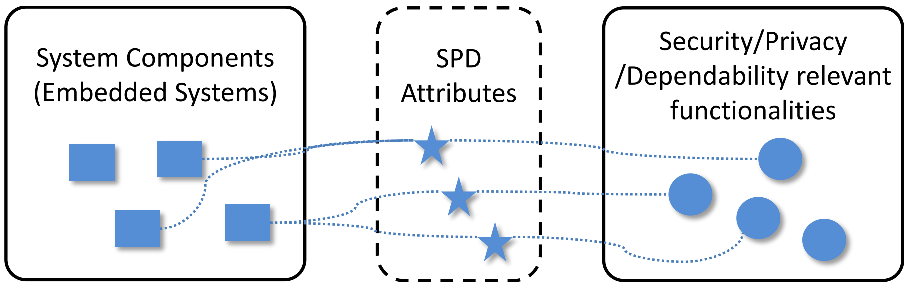

System Components (Embedded Systems)
SPD Attributes
Security/Privacy /Dependability relevant functionalities

Figure 3.1: Proposed approach to model SPD for ES

In particular the metamodels that describes relations has been built by taking into account the atomic attributes that are impacted in SPD context and mapping them in these two worlds.

The whole pSHIELD meta-model is depicted in Figure 3.2, and the prototype is available in AD4.

For all the three models, a short justification and a description are provided in the following paragraphs.

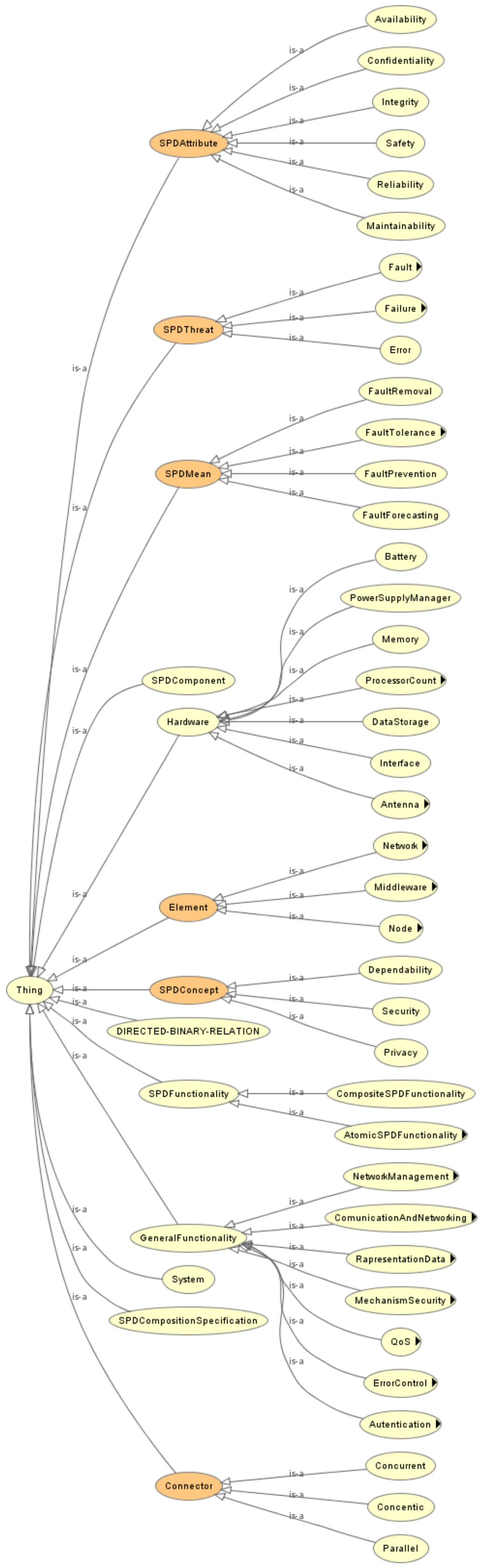


Figure 3.2: pSHIELD meta-model

3.1.1.1 Structural System meta-model

SHIELD is a framework composed by the interaction of dozens of interconnected Embedded Systems: they constitute the "physical world". This world is made by hardware and software components, mapped on three layers: Node (the devices), Network (the interaction between devices) and Middleware (the software services that make the devices run). The first semantic model captures this concept.

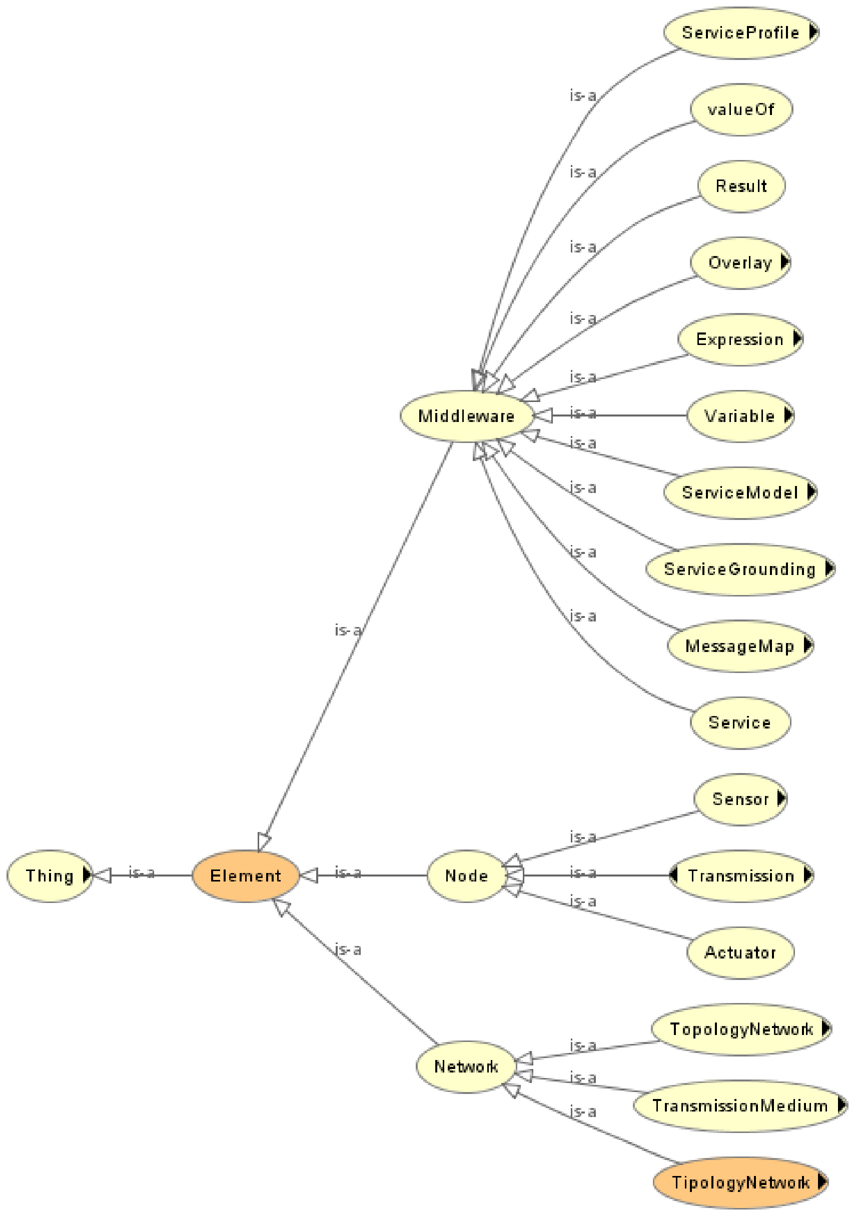


Figure 3.3: Structural Ontology

3.1.1.1.1 pSHIELD Node Model

The structural ontology is the easier to model, because it is a simple description of the Embedded System component. It contains the hardware components and basic functionalities provided by the individual element and the related attribute, all in an SPD relevant environment. For example a node, in a first simplification, is composed by a memory a CPU, a battery and a transmission antenna; furthermore the CPU is characterized by the frequency and bit length and, in SPD relevant context, the possibility of performing hardware cryptography. By doing so, all the components constituting a complex system can be represented. The elements that constitute a node are represented in Figure 3.4 and Figure 3.5.

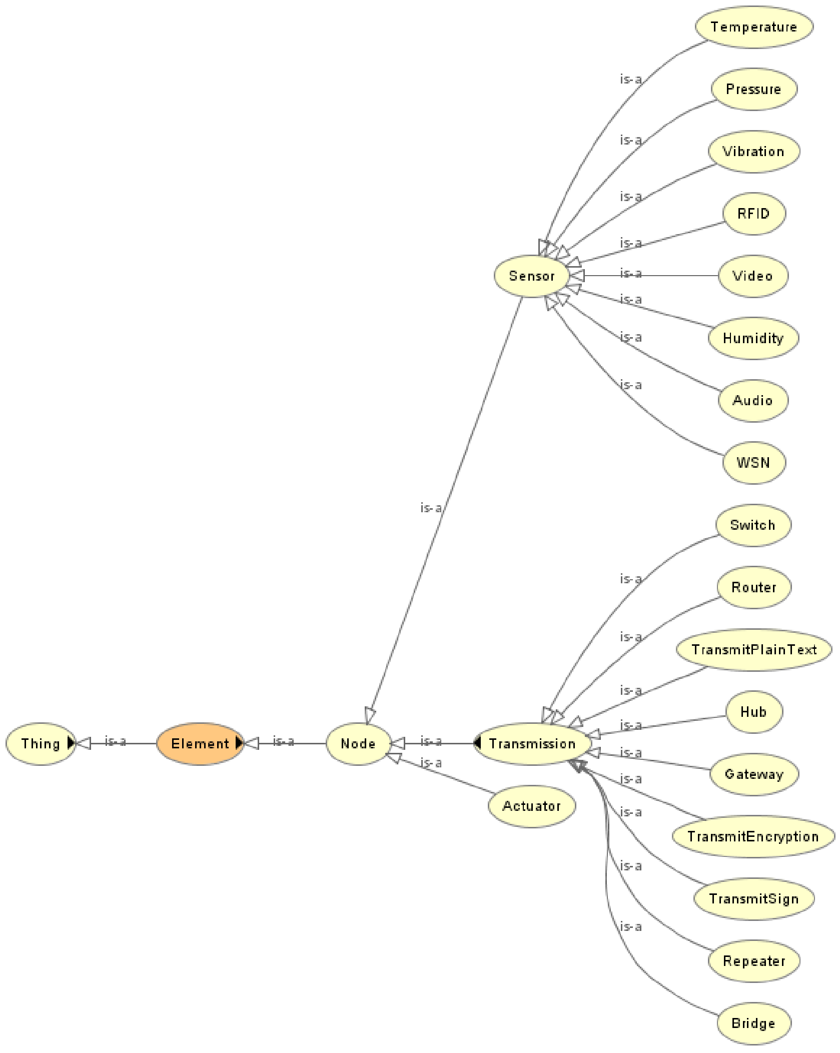


Figure 3.4: pSHIELD Node Model

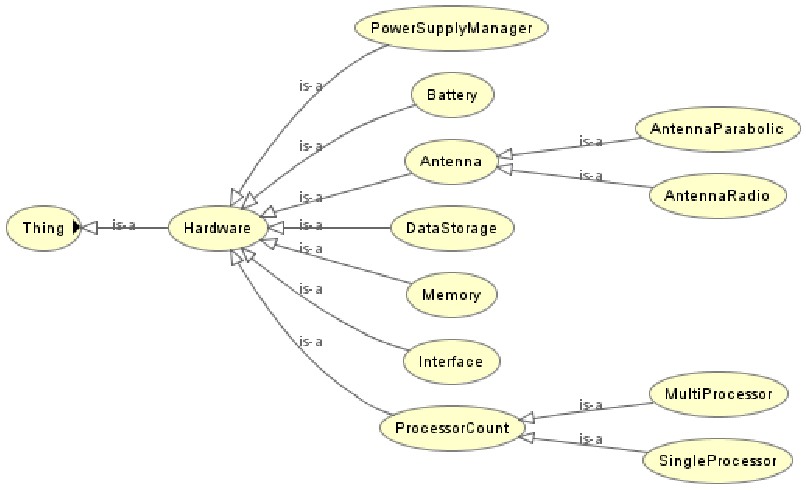


Figure 3.5: Node hardware ontology

3.1.1.1.2 pSHIELD Network Model

The network model was trivial to derive, since the only relevant information are about the topology, the transmission medium and the typology.

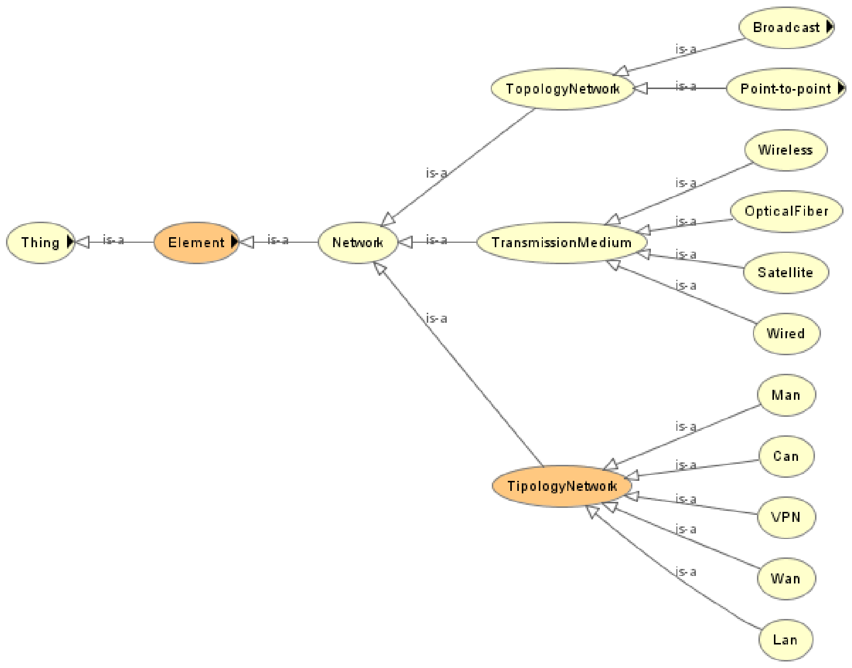


Figure 3.6 pSHIELD Network Model

3.1.1.1.3 pSHIELD Middleware and Overlay Model

The model for the Middleware services is the standard OWL-S to describe services. This choice has been done to maximize interoperability.

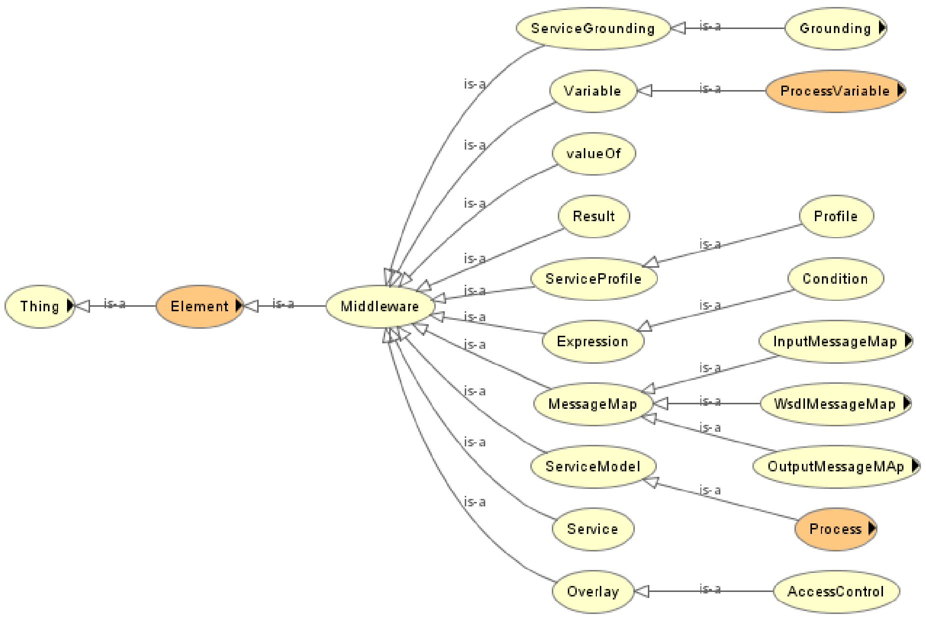


Figure 3.7: pSHIELD Middleware Model

3.1.1.2 SPD Functionalities meta-model

The SPD functionalities were modelled as composable atomic components.

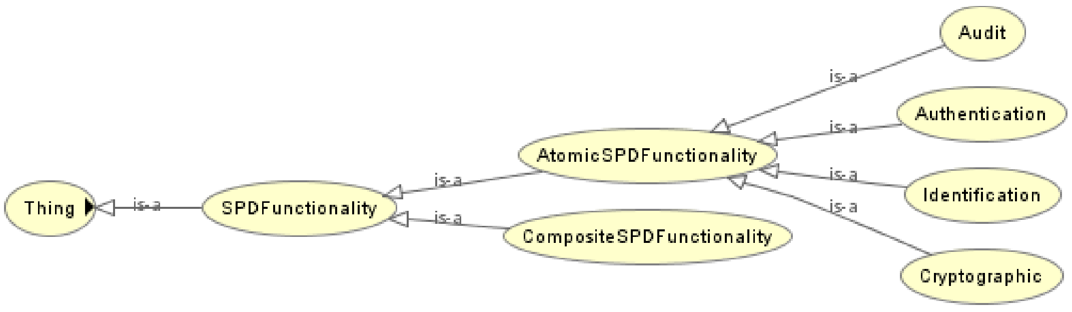


Figure 3.8: SPD Functionality

A `SPD Functionality` can be an `Atomic SPD Functionality` or a `Composite SPD Functionality`: the latter embodies a composite pattern (from a functional point of view it acts as a `SPD Functionality`) and represents an aggregation of other `SPD Functionality` (possibly composite) by means of a `Connector`.

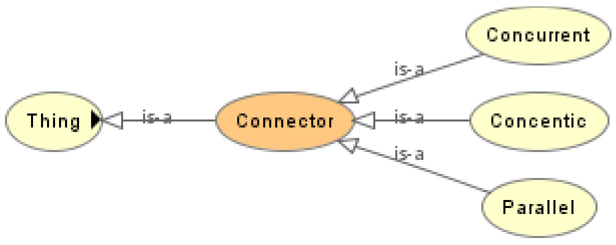


Figure 3.9: Connector

The `Connector` stands for a mean of aggregation, and specifies:

- one of the identified patterns: A connector provides a specification of the structure of the composition, by means of basic "control constructs" (whose names are reminiscent of control structures in programming languages). Specific instances of connector implement the conceived aggregations of pSHIELD SPD Metrics
- a `SPDCompositionSpecification`: a connector offers an analytical specification of the algorithm that composes the SPD status values of contributing SPD functionalities into the overall SPD status value.

Composition is modelled after analogous Composite Processes in OWL for services (OWL-S). The following analogies have been drawn:

Table 3.1: SPD Composition modelling

SPD Composition	Description	Analogous OWL-S Control Construct
Concentric	The SPD functionalities are composed serially	Sequence
Concurrent	The SPD functionalities are composed separately on different assets	Split
Parallel	The SPD functionalities protect at the same time the same assets	Choice

A `SPD Functionality` can be measured by a `SPDStatus` that ultimately assesses the overall functionality of the ES from the viewpoint of SPD.

3.1.1.3 SPD Attributes

SPD concepts are described in terms of *attributes* (that assess them), *threat* (that affect them) and *means* (that increase them).

The attributes are derived from the SPD taxonomy identified in the pilot phase (see Figure 3.10).

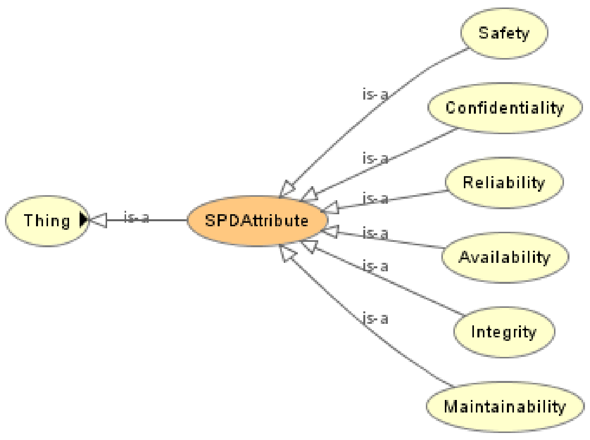


Figure 3.10: SPD Attributes

Following the Common Criteria approach, the SPD attributes are:

- Menaced by SPD Threats (error, failure, …)
- Improved by SPD Means (fault forecasting, tolerance, ecc.)

This is depicted in the following Figure 3.11 and Figure 3.12.

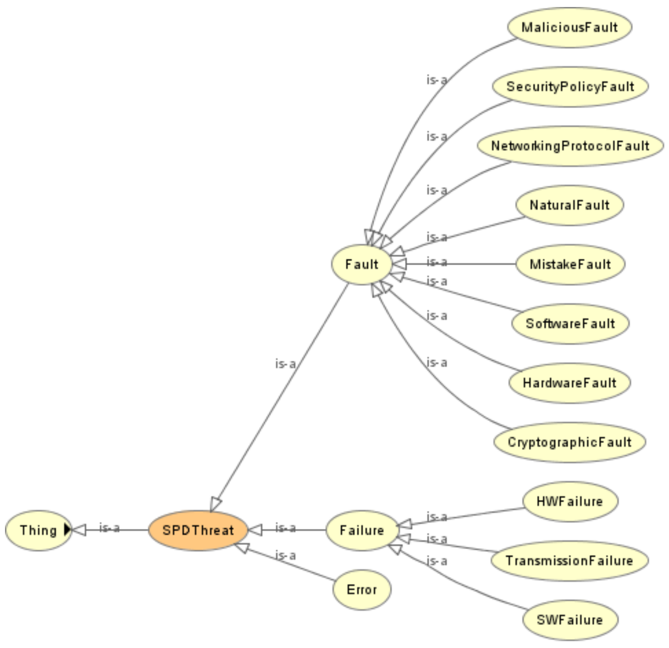


Figure 3.11: SPD Threat

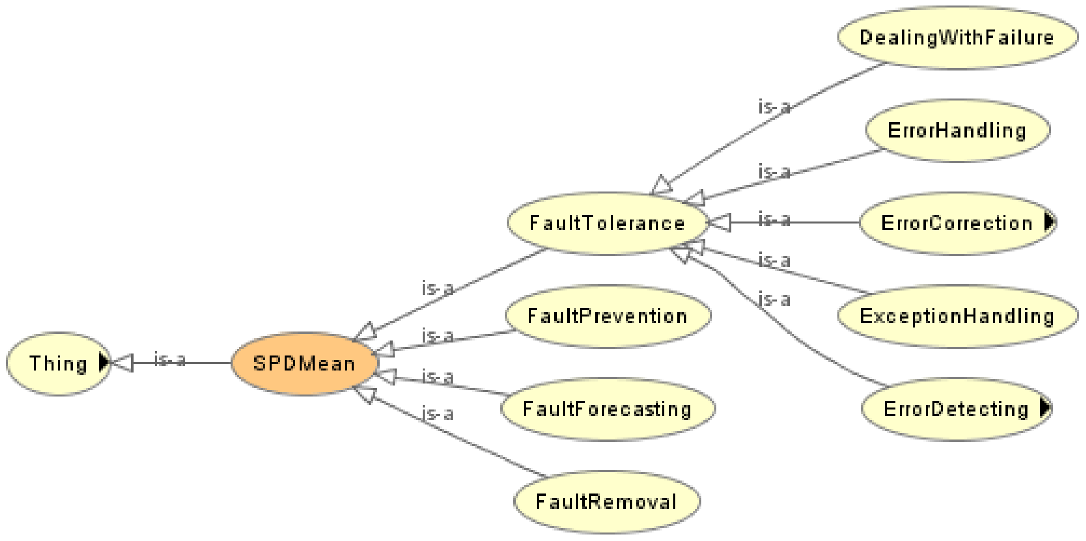


Figure 3.12: SPD Mean

3.1.1.4 Remarks

To sum UP, the following metamodels have been derived in the pilot phase:

- A section to represent *system's components*
- A section to represent *functional properties*
- A section to represent *SPD relevant information:* attributes, threats, means of mitigation

Plus:

- Attributes to identify *relations* between system and functionalities
- Attributes to quantify *SPD level*

All the presented metamodels are linked by relations that allow a complete knowledge of the environment, according to this logical chain:

1. The pSHIELD System is composed by Node, Network and Middleware elements
2. These elements are made by real hardware components and realizes some functionalities
3. Some of these components can be considered as SPD Component as well as some of these functionalities realise SPD Functionalities
4. SPD Functionalities can be composed
5. SPD Functionalities impact SPD Attributes
6. SPD Attributes are affected by SPD Threats
7. SPD Threats can be improved by SPD Means

The logical chain is depicted also in Figure 3.13, where the dotted lines represent the separation between the system and the user: the user drives the composability with respect to the desired SPD level, and following the paths, all the systems elements are composed (and quantified) consequently.

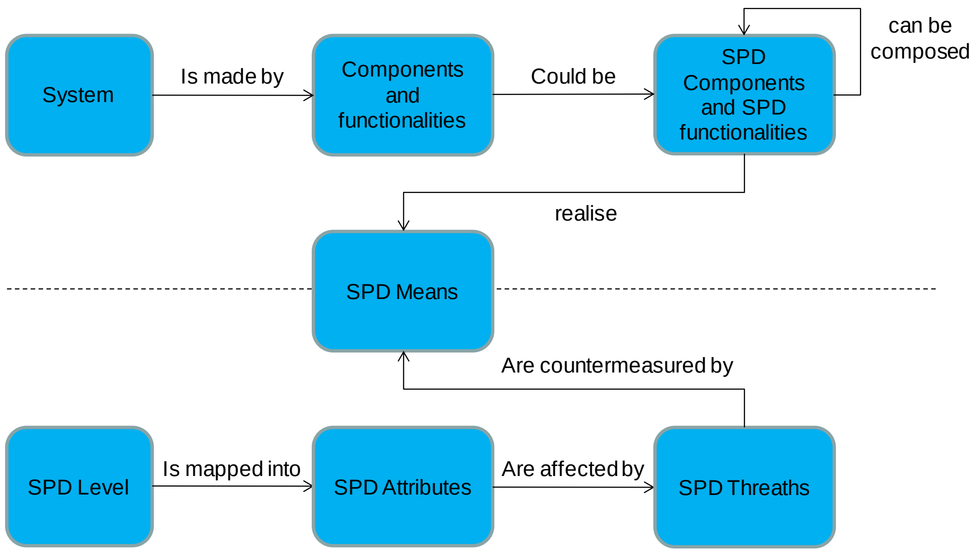


Figure 3.13: pSHIELD ontology logical chain

In conclusion, the assessment of this activity is reported below:

Table 3.2: Procedure to define the pSHIELD semantic model – ASSESSMENT

Procedure to define the pSHIELD semantic model - ASSESSMENT	
Limits of this approach to be overcome	This result was more than suitable for the demonstration purposes, where a reduced scenario was addressed and a small set of components and models had to be modelled. However some limits have been identified: • Too much queries have to be performed to go through the whole chain: this could be a problem in scenarios with hundreds or thousands of components and models both for memory and time constraints. • The metamodels associated to the physical system are too specific, so a huge initial effort is needed to translate, into the ontology, the description of the various components. • The metamodels associated to the physical system are not scalable, since the development of a new technology should be manually updated into each component
Positive aspects to be preserved	The part of the modelling procedure derived from the Common Criteria (SPD Level-Attributes-Threats-Means) is linear, simple and efficient: it allows to establish a link between the user needs and the system behaviour, but keeps them decoupled. This logical chain should be preserved also in the prosecution of the research.

3.1.2 pSHIELD semantic technology

For the implementation of the pSHIELD ontology, the OWL language has been selected, after a careful evaluation that took into account expressiveness and computational complexity (or memory usage).

The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with formal semantics. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

The basic reasons for decision to use of OWL for modelling in pSHIELD are:

1. OWL extends all other languages like XML, RDF, and RDF-S. Actually, OWL has been developed on top of the existing XML and RDF standards, which did not appear adequate for achieving efficient semantic interoperability.

   a) E.g. in XML and XML Schema same term may be used with different meaning in different contexts, and different terms may be used for items that have the same meaning.

   b) E.g. RDF and RDF-S address some problem by allowing simple semantics to be associated with identifiers. With RDFS, one can define classes that may have multiple subclasses and super classes, and can define properties, which may have sub-properties, domains, and ranges. However, in order to achieve interoperation between numerous, autonomously developed and managed schemas, richer semantics are needed, like disjoints and cardinality of relations.

2. OWL adds more vocabulary for describing properties and classes, relations between classes, cardinality, equality, richer typing of properties, characteristics of properties and enumerated classes, and all available in three increasingly expressive and increasingly complex sublanguages (Lite, DL, Full) designed for use by specific communities of implementers and users.

3. OWL is well-known widely used open W3C standard with very good support and promising potential and real usage in several industry applications.

4. OWL has wide support of modelling tools, platforms, and reasons.

5. Previous languages could express (in most cases) the same things, but for some of them OWL provide direct solution by a predefined type of predicates.

6. There are several well-known mechanisms for expressing OWL-Lite and OWL-DL ontologies to stay on decidable level, where Description Logic (DL) could be used correctly.

7. OWL language has proved its potential to use for modelling of semantic interoperability in several middleware-based applications and domains.

In pSHIELD the same OWL-based framework can be used for representation of context, device descriptions (capabilities), descriptions of middleware components, services, security aspects, with several specific goals such as:

1. Semantic reasoning based on ontology model may carry out a reconciliation of heterogeneous formats of parameters exchanged between different layers (also suitable for interaction with legacy agents).

2. The semantic characterization of the behavioural aspect of components makes it suitable for an agent to determine “what the service does”.

3. The semantic characterization of the composition of functionalities and of the relations among them makes it suitable for an agent to reason about SPD metrics of the current configuration and - if needed - to carry out reconfigurations of the system at run-time, by means of rule-based combination / composition of components and SPD technologies, in order to achieve the new intended values for SPD metrics.

The ontology has many merits, of which the most notable are the excellent extensibility, and high expression power. Many systems in the “ubiquitous” and embedded environments are being developed using DL-based ontologies and used with DL-based reasoning. Usually, ontologies are used for modelling context that the systems should collect and analyse. A pure DL-based approach, however, has certain limitations in a context environment. OWL and other ontology languages based on Description Logic cannot properly handle rules expressed in Horn-Logic. Hence, to ensure syntactic and semantic interoperability on device level (e.g. “low-level” ontologies), *SWRL* (Semantic Web Rule Language) or *Jena* can be used for expressing rules.

The final implementation of the pSHIELD OWL was an XML file, easy to be imported by the middleware and semantic parses. The full XML code of the pSHIELD ontology can be found in AD4 (a screenshot is provided in Figure 3.14).

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF
[…]
<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1300273978.owl#"
     xml:base="http://www.owl-ontologies.com/Ontology1300273978.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
     xmlns:Ontology1300273978="http://www.owl-ontologies.com/Ontology1300273978.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:TCP="&Ontology1300273978;TCP/">
    <owl:Ontology rdf:about="">
        <owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/protege"/>
    </owl:Ontology>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://www.owl-ontologies.com/2005/08/07/xsp.owl#minExclusive -->
    <owl:ObjectProperty rdf:about="&xsp;minExclusive">
        <rdfs:domain rdf:resource="&rdfs;Datatype"/>
    </owl:ObjectProperty>

    <!-- http://www.owl-ontologies.com/Ontology1300273978.owl#HasAutorization -->
    <owl:ObjectProperty rdf:about="#HasAutorization"/>
    […]
```

Figure 3.14: pSHIELD OWL (XML File)

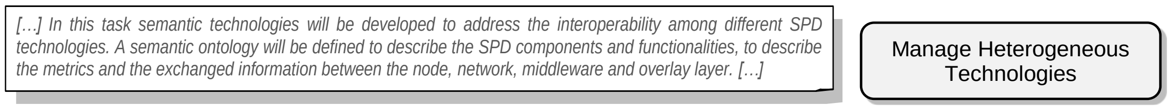
The assessment of this technology is reported in the following:

Table 3.3: pSHIELD semantic technology – ASSESSMENT

pSHIELD semantic technology - ASSESSMENT	
Limits of this approach to be overcome	The OWL language used in pSHIELD showed to be adequate for the railways demonstration purposes, where discrete computational capabilities were available; however it could not fit all the possible application, due to the reduced memory and computational capabilities of ESs. A new way of representing the ontology and the metamodels is needed.
Positive aspects to be preserved	The translation of the model into an XML file is scalable and versatile, so it will be for sure a constant also in the technological choices for the nSHIELD project.

3.2 nSHIELD potential investigations

NOTE: Starting from this paragraph and for all the others related to nSHIELD investigation, the text from the Technical Annex (shadowed boxes), as well as the user/scenario needs (rounded boxes), will be reported as justification of the technological investigations.



[…] In this task semantic technologies will be developed to address the interoperability among different SPD technologies. A semantic ontology will be defined to describe the SPD components and functionalities, to describe the metrics and the exchanged information between the node, network, middleware and overlay layer. […]

Manage Heterogeneous Technologies

Figure 3.15: Technical Annex text box (left) and user/scenario bullet (right)

3.2.1 Proposed procedure to define the SHIELD semantic model

[…] In this task semantic technologies will be developed to address the interoperability among different SPD technologies. A semantic ontology will be defined to describe the SPD components and functionalities, to describe the metrics and the exchanged information between the node, network, middleware and overlay layer. […]

Manage Heterogeneous Technologies

The biggest limitation of the pSHIELD approach (Table 3.2) was the high number of models and their strict relation with the physical system (i.e. great effort to build a knowledge base that, at last, is not scalable). The only way to overcome this complexity is by simplifying the information through abstraction: only a subset of information has to be selected, applicable to all the devices and components, and then they have to be represented in the same way, no matter whether they are software or hardware technologies, middleware, node or network component. A uniform, simple and abstract model thus enables the interoperability, for composability purposes, between SPD technologies. But what happens to the non-uniform information?

Technology independent abstraction

There are, indeed, a plenty of information in the system description that cannot be abstracted or represented in a uniform way (metrics, measurements, topology, architectural dependencies, …); for that reason, in order to preserve the completeness of the knowledge representation, a second model should be added to the former, containing all those information related to the specific application scenario or underlying technology that cannot be abstracted.

A decoupling procedure will be adopted to translate the three pSHIELD metamodels into two new SHIELD metamodels:

- a technology independent one, plus
- a domain library.

This is the only reasonable solution, since it is not feasible to imagine a unique representation of a domain as complex as the Embedded Systems one, mainly due to the scalability and expressiveness problems identified in the assessment.

The first model will feed the composability computation, while the second one will drive the composability implementation.

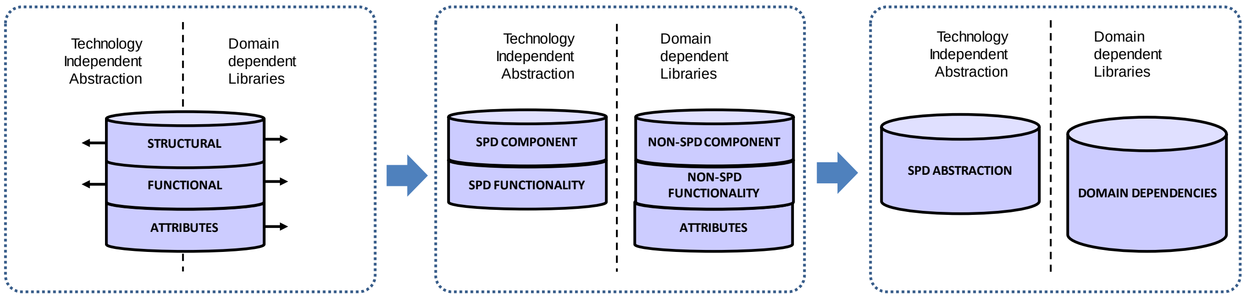

Technology Independent Abstraction
Domain dependent Libraries
STRUCTURAL
FUNCTIONAL
ATTRIBUTES
Technology Independent Abstraction
Domain dependent Libraries
SPD COMPONENT
SPD FUNCTIONALITY
NON-SPD COMPONENT
NON-SPD FUNCTIONALITY
ATTRIBUTES
Technology Independent Abstraction
Domain dependent Libraries
SPD ABSTRACTION
DOMAIN DEPENDENCIES

Figure 3.16 Decoupling of the pSHIELD ontology

The decoupling process is depicted in Figure 3.16. The three original metamodels are split in two halves:

- the model describing the structure of the component, as well as the model describing the SPD functionalities, are split into SPD and NON-SPD models[3], while
- the attributes models is considered as domain specific model.

This is in line with what is expected from the Technical Annex.

[…] In order to build the target functionalities a modular approach will be followed by partition the features of the middleware in abstract SPD modules. Each module will be a collection of conceptually similar functions that provide services to other modules or layers: the SPD modules will be characterized to be dynamically composable. […]

In the following Figure 3.17, a better explanation of the SHIELD semantic model is provided. In particular the components identified at the three canonical layers (node, network and middleware) are translated into a unique, atomic, uniform element named *Abstract SPD Functionality* that identifies:

- A SPD functionality
- The associated SPD value
- The relations with other SPD functionalities
- The constraints that limit the composition possibilities.

All the remaining information are transferred into the domain knowledge bases, including deployment/ architectural dependencies, metrics evaluation, policies, etc.

[3] NON-SPD information are mainly functional information

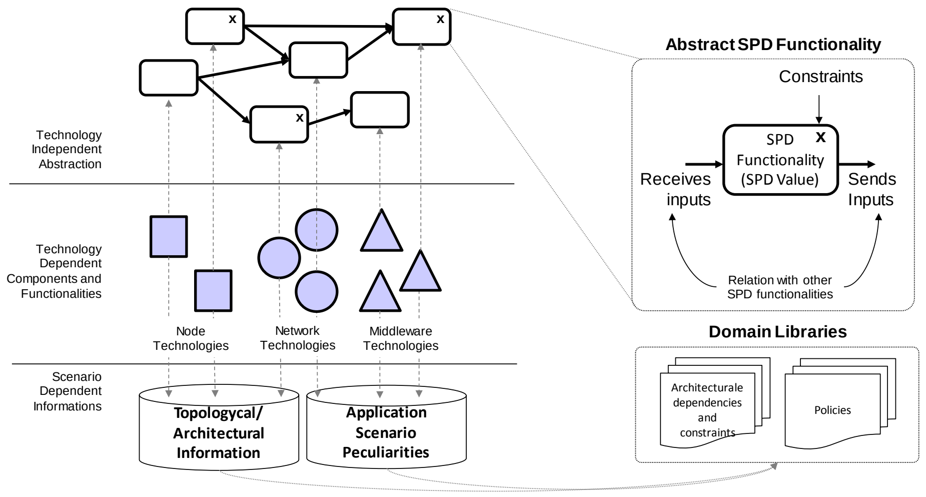

Technology Independent Abstraction
Technology Dependent Components and Functionalities
Scenario Dependent Informations
Node Technologies
Network Technologies
Middleware Technologies
Topologycal/ Architectural Information
Application Scenario Peculiarities
Abstract SPD Functionality
Constraints
SPD Functionality (SPD Value)
Receives inputs
Sends Inputs
Relation with other SPD functionalities
Domain Libraries
Architecturale dependencies and constraints
Policies

Figure 3.17: SHIELD semantic models

The logical chain identified in the pilot phase is still valid (and simplified, as needed from the assessment). It is depicted in Figure 3.18.

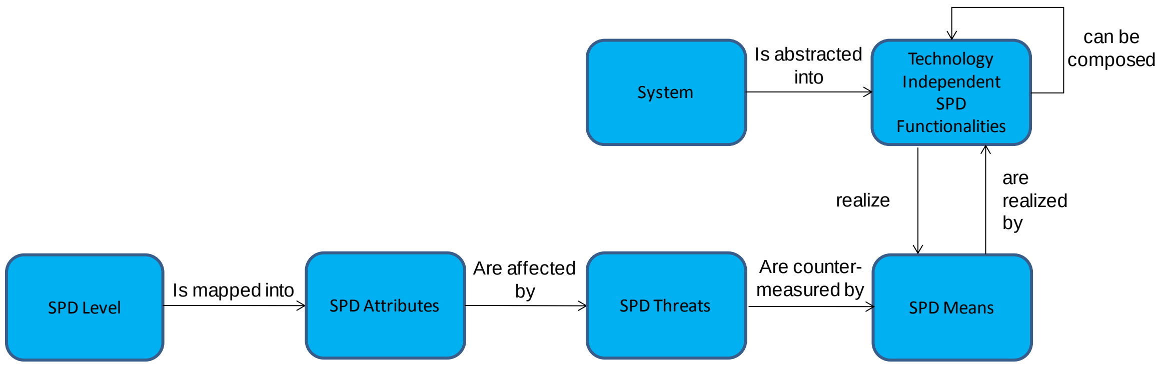


Figure 3.18: SHIELD ontology logical chain

As an example of the new semantic abstraction, we will consider *cyphering* (see Figure 3.19). This can be done either by hardware or software modules, but their abstract representation will be the almost same: a box named cyphering. Then, the cyphering performed by HW, will have an input relation with, for example, the power management functionality, since the availability of power resources could affect the possibility of performing cyphering or not; while the cyphering performed by software module doesn't have input relations with other functionalities.

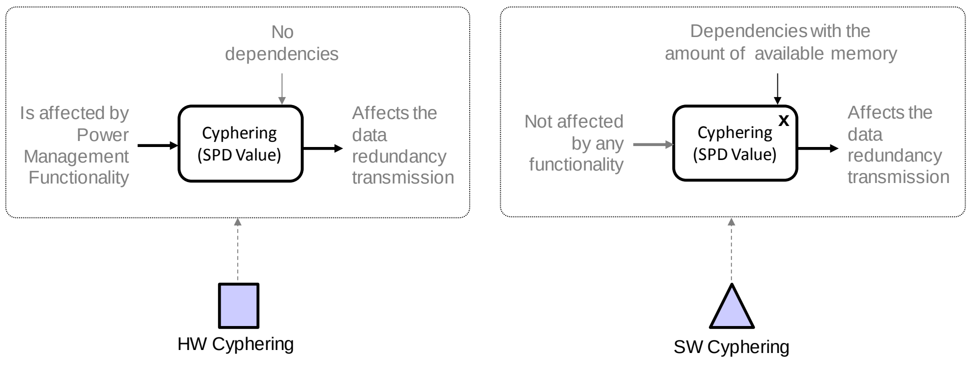

No dependencies
Is affected by Power Management Functionality
Cyphering (SPD Value)
Affects the data redundancy transmission
HW Cyphering
Dependencies with the amount of available memory
Not affected by any functionality
Cyphering (SPD Value)
Affects the data redundancy transmission
SW Cyphering

Figure 3.19: Example of technology abstraction

Secondly, all the functionalities affect the network data redundancy module, since they may introduce overhead and increase the bandwidth occupied by the transmitted data.

Finally, the HW cyphering doesn't have architectural dependencies (if there is a CPU able to perform cyphering, then it can do it), while the SW cyphering has an architectural (NON-SPD) constraint related to the memory usage of the cyphering routine.

This constraint is stored in the domain knowledge base, together with the SPD value of the component that depends on several (domain related) aspects. For example a cyphering module that sends data from two components of a camera for railways surveillance is affected by physical menaces, while the same module installed on a flying UAV is very difficult to be physically attacked.

Further semantic studies lie on the above mentioned models. A couple are reported in the following paragraphs.

3.2.1.1 An example of technological abstraction: the algebra of connectors

The main benefit of the technology independent abstraction is the possibility to define algebras to model the composability of SPD functionalities. In fact, given a uniform model (like the one defined in Figure 3.17), there are many ways to model their relations (with the perspective of feeding the control algorithm at overlay level). In particular a promising research field has been identified, based on the theory of connectors (i.e. the possibility of modelling interconnected uniform components) developed in computer science.

In recent years, computer science researchers shift their focus from traditional isolated computing system to massively distributed communication ones. The modern communication technologies have entailed the conceptual separation between *coordination* and *computation* in distributed computing system. This separation should be noted in different levels of abstraction, such as architecture, software, process, etc., where concepts like heterogeneity and reusability require modular specifications, theories and models.

In this scenario, the concept of *connector* has emerged to interact and interoperate small functionalities and services that are separately developed. Connectors are the *glue code* that takes care of all those aspects.

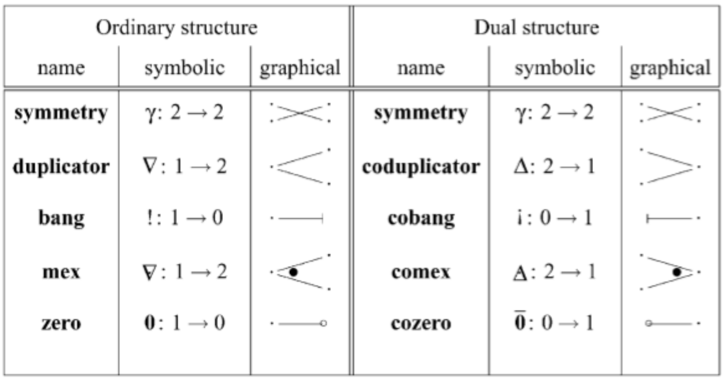
To analyse interconnected and distributed systems is crucial to have good mathematical foundations. Several connector categories have been studied in the literature and all bring out the common definition of connector: "a component that mediates the interaction of other computational components and connectors" [83]. Typically, connectors have been studied within two important frameworks for system modelling *categorical* and *algebraic approaches* (see for example [84] for categorical approach and [85],[86] for algebraic approach).

In the categorical approach, the systems are represented with the objects in a category and the sub system and the relations are developed through morphisms.

In the algebraic approach, systems are implemented through a suitable algebra. This has only basic components that can be used to achieve the other operators and complex systems.

Several papers [83], [87], [88] offer different algebraic approaches with simple basic components.

For example, in [83] the authors present a basic algebra of stateless connectors using syntax with only five basic connectors and its duals (see Figure 3.20).



Ordinary structure			Dual structure		
name	symbolic	graphical	name	symbolic	graphical
symmetry	$\gamma: 2 \rightarrow 2$		**symmetry**	$\gamma: 2 \rightarrow 2$	
duplicator	$\nabla: 1 \rightarrow 2$		**coduplicator**	$\Delta: 2 \rightarrow 1$	
bang	$!: 1 \rightarrow 0$		**cobang**	$¡: 0 \rightarrow 1$	
mex	$\nabla: 1 \rightarrow 2$		**comex**	$\Delta: 2 \rightarrow 1$	
zero	$\mathbf{0}: 1 \rightarrow 0$		**cozero**	$\overline{\mathbf{0}}: 0 \rightarrow 1$	

Figure 3.20: Example of basic semantic in [1]

Moreover, in [87] the authors define another semantic with six primitive, but with different approach of the basic elements. Figure 3.21 shows the primitive of this algebra.

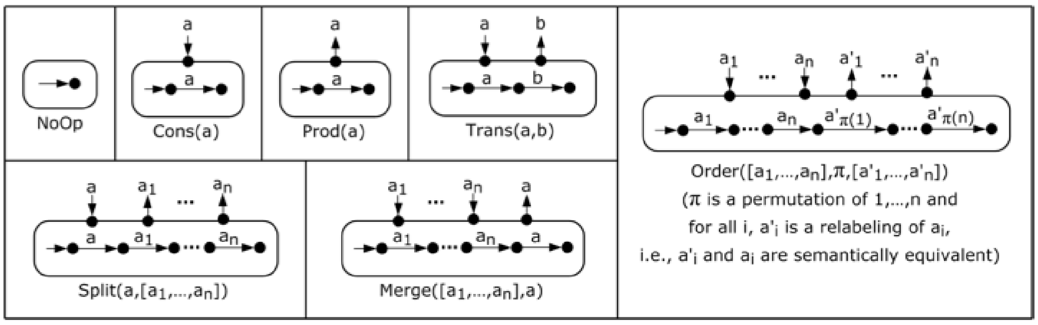

NoOp
Cons(a)
Prod(a)
Trans(a,b)
Order([a1,...,an],π,[a'1,...,a'n])
(π is a permutation of 1,...,n and
for all i, a'i is a relabeling of ai,
i.e., a'i and ai are semantically equivalent)
Split(a,[a1,...,an])
Merge([a1,...,an],a)

Figure 3.21: Example of basic semantic in [12]

All of these aforementioned documents establish that there is not a single algebra, but this is developed *ad hoc* for a single scenario aiming to model systems where multiple actions can be executed at each time, either independently or synchronized.

The main benefit of the development of connector algebra, or at least composition formalism, is the possibility to evaluate the individual contribution of the SPD component to the overall SPD. This approach should provide the SPD quantification framework, since the SPD quantification/evaluation rules or algebra is in charge of WP2 and the framework should be independent from the specific quantification/evaluation process.

Assess overall security

3.2.1.2 An example of domain knowledge base: IDS Ontologies

The technology independent ontology is mainly used for composability purposes, with the enrichment of domain knowledge base. However this domain related knowledge base can be used also to perform other SPD relevant activities that enrich the SHIELD behaviours. An interesting application that will be analysed in the nSHIELD project is the possibility of using ontologies to perform intrusion detection (eve in this task is already in charge to the middleware core services).

Intrusion detection systems (IDS) can be defined as a set of different scanners that monitor the activities of an information system looking for malicious actions. An IDS is not an antivirus designed to detect malware or a first line barrier like firewall, it is a detection system that identifies anomalous activities, alerts about them and optionally takes reactive actions to sub sane them.

We can classify the different kind of existing IDS based on the following criteria:

- Host-based versus Network-based: The IDS agents can be installed inside hosts, and they run apart from the normal functionalities of host, monitoring the activities inside the host. These are called Host intrusion detection system (HIDS).

Instead in Network intrusion detection system (NIDS), the agents are installed in special devices and monitor the network traffic. Usually these devices are completely transparent for the network because they do not answer to any link layer or network layer addresses, i.e., they operate as passive devices. If they need to communicate with reporting or management systems, they usually have a second link to a different network (the management network).

- Centralized versus Decentralized: In centralized IDS there is an agent that collects and analyses all the anomalous activities of the whole system and sends alerts or takes actions depending on the result the analysis.

  On the other hand, decentralized IDS are composed of several agents that run independently and collect and analyse their own anomalous activities, taking their own measures. This type of IDS is more resistant and versatile, but also more complicated to configure and maintain.

- Type of analysis used to detect anomalous activities: In misuse-based analysis the IDS agent compares the data from the monitored system with known malicious patterns stored in its attack database. If it finds a match it will raise an alarm (an optionally adopt a counter measure). On the other hand, in anomaly-based analysis the IDS agent compares the monitored system activity with the normal behaviour that it is supposed to have (this normal behaviour is usually modelled beforehand).

In the scope of the project, the IDS will be the first safety barrier for possible attacks against the system, warning of possible attacks to maintain reliability and availability of the network.

In the recent years, some people have started to develop methods which separate the design process into a high level and a low level phase, helping the developer to design more powerful network intrusion detection.

As Hung et al. [76] affirm, these approaches define methodologies for designing an intrusion detection application which meets the end-user requirements. However, they do not express the modelling of the intrusion detection application in terms of the domain of interest. They posit that it is important not only to corporate the terminology of a domain but also to make sure that domain expert with this terminology of the domain can fully exploit his/her domain expertise for designing his/her intrusion detection application.

Undercoffer et al. [77,78] were the first to propose ontologies for intrusion detection. From the point of taxonomy, the intrusion detection can be considered as possessing many characters and classifications and it needs a language that describes instances of that taxonomy. The language is paramount to the effectiveness of the intrusion detection system because information regarding an attack or intrusion needs to be intelligently conveyed, especially in distributed environments.

After the initial proposal from Undercoffer et al., several authors have proposed different or more detailed ontologies. We can cite the following ones as especially relevant in the context of the project:

- The previously cited Hung et al. [76]
- The work from Abdoli and Kahani[79]
- The work from Isaza et al[80, 81]. These authors not only define an ontology for IDSes but they also define a methodology [82] called METHONTOLOGY to develop the ontology.

They affirm that they use METHONTOLOG because in different studies it is considered one of the most mature methodologies that seek to follow the life cycle of the software proposed in the IEEE 1074 standard, which is recommended by the Foundation for Intelligent Physical Agents (FIPA). The methodology not only incorpores the description of the attack taxonomy, but also axioms and rules describing the attacks.

3.2.2 Proposed SHIELD semantic technologies

[…] The outcome of this task will be a lightweight common semantic languages derived by standard ones (OWL) in order to be easily processed in the embedded system world where the processing unit are limited in power and resources. The semantic ontology will be part of the prototypes delivered by WP5. […]

The limits of the OWL used in the pilot phase have been identified in Table 3.3, so new languages should be examined to ease i) the design of the SHIELD metamodels and ii) the implementation of the SHIELD metamodels into demonstrators.

Two candidate technologies will be studied, to face these issues.

3.2.2.1 MARTE DAM UML

pSHIELD ontology was developed using ER diagrams translated into OWL files with Protégé software (see Figure 3.22).

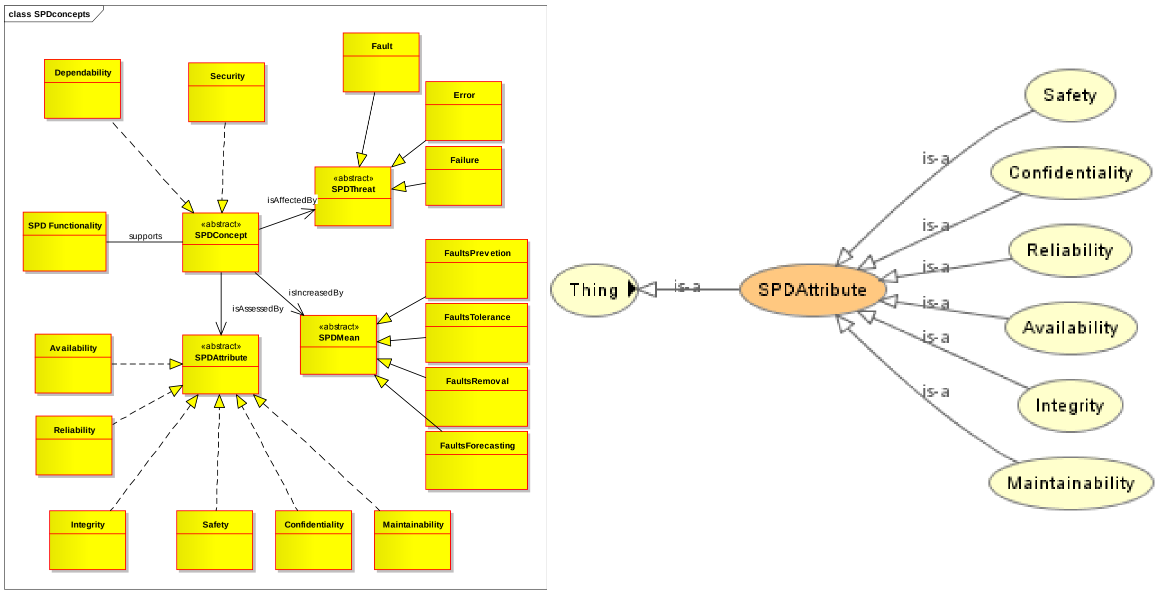


Figure 3.22: pSHIELD semantic model design from ER to OWL

This is not user friendly enough, so the adoption of UML as modelling language of the SHIELD semantic model will be considered. In particular there is a specific profile, named ***MARTE***[4] ***DAM***[5] ***UML***, that enriches the UML language with the possibility of modelling information and attributes related to the embedded systems domain in safety-critical application and in dependability analysis (see for an example).

In is interesting to underline that the DAM profiles foresees specific constructs that follows the logical chain defined in the SHIELD project (Figure 3.18), since it includes construct to define menaces, threats, faults and their attributes.

4 Modeling and Analysis of Real-Time and Embedded systems

5 Dependability Analysis and Modeling

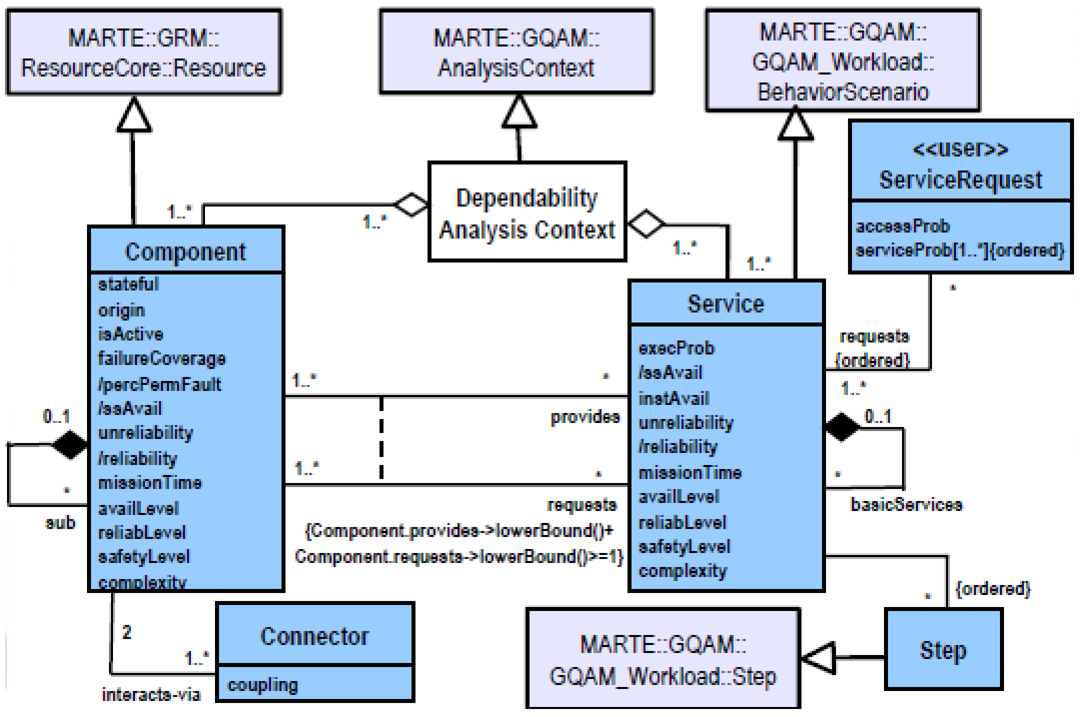


Figure 3.23: MARTE DAM UML example

3.2.2.2 JSON

The versatility of XML as language to translate the SHIELD semantic models has been widely proved in the pilot phase, so XML will be the basis also of the nSHIELD project. However a second language will be investigated, specific for lightweight application. The following text is reported from the official JSON initiative[6]

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

[6] http://www.json.org/

An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by **:** (colon) and the name/value pairs are separated by **,** (comma).

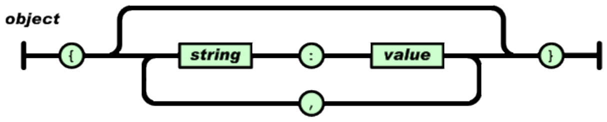


Figure 3.24: JSON object

An *array* is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).

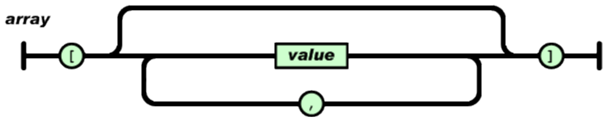


Figure 3.25: JSON array

A *value* can be a *string* in double quotes, or a *number*, or true or false or null, or an *object* or an *array*. These structures can be nested.

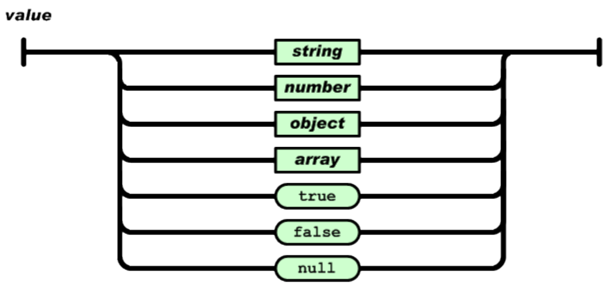


Figure 3.26: JSON value

A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.

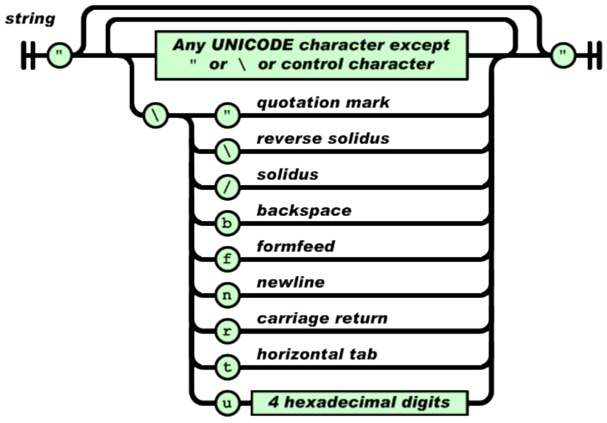


Figure 3.27: JSON string

A *number* is very much like a C or Java number, except that the octal and hexadecimal formats are not used.

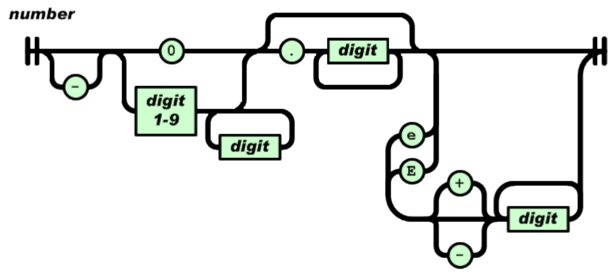


Figure 3.28 JSON number

Whitespace can be inserted between any pair of tokens. Excepting a few encoding details, which completely describe the language.

The main usage of these languages is, for example, in embedded system domain, where low powered devices (like the Arduino platform[7]) can exchange information in an efficient way with this lightweight semantic.

[7] www.arduino.cc

4 SHIELD middleware core SPD services assessment

4.1 pSHIELD results and adopted technologies

In the pilot project it has been chosen to implement a reduced but significant set of core SPD services as enabling of the pSHIELD middleware behaviour. The core SPD services aim to provide a SPD middleware environment to actuate the decisions taken by the SHIELD Overlay and to monitor the Node, Network and Middleware SPD functionalities of the Embedded System Devices under the SHIELD Middleware Adapter control. The selected core SPD services are (see Figure 3.1):

- service discovery;
- service composition;
- service orchestration.

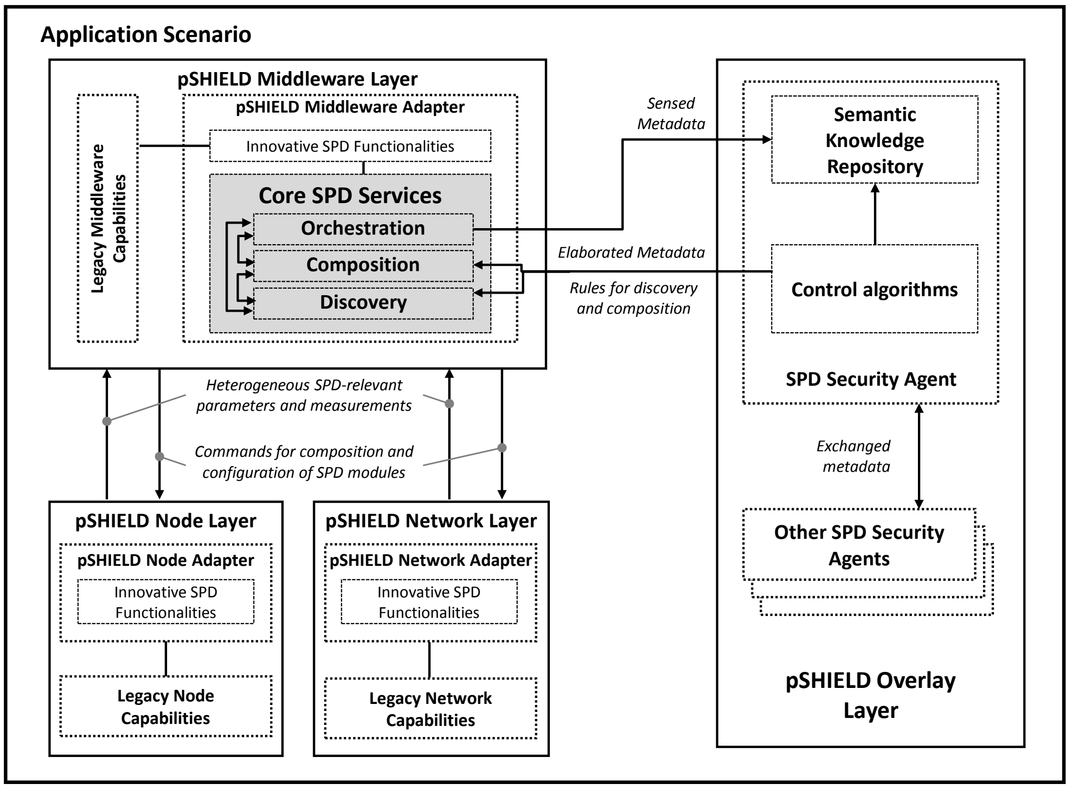

Application Scenario
pSHIELD Middleware Layer
pSHIELD Middleware Adapter
Legacy Middleware Capabilities
Innovative SPD Functionalities
Core SPD Services
Orchestration
Composition
Discovery
Sensed Metadata
Elaborated Metadata
Rules for discovery and composition
Semantic Knowledge Repository
Control algorithms
SPD Security Agent
Heterogeneous SPD-relevant parameters and measurements
Commands for composition and configuration of SPD modules
Exchanged metadata
Other SPD Security Agents
pSHIELD Node Layer
pSHIELD Node Adapter
Innovative SPD Functionalities
Legacy Node Capabilities
pSHIELD Network Layer
pSHIELD Network Adapter
Innovative SPD Functionalities
Legacy Network Capabilities
pSHIELD Overlay Layer

Figure 4.1: Core SPD services selected for the pilot project

4.1.1 pSHIELD discovery engine

Service discovery is the service that allows any SHIELD Middleware Adapter to discover the available SPD functionalities and services over heterogeneous environment, networks and technologies that are achievable by the SHIELD Embedded System Device where it is running. Indeed the SHIELD secure service discovery uses a variety of discovery protocols (such as SLP[8], SSDP[9], NDP[10], DNS[11], SDP[12],

[8] IETF Service Location Protocol V2 - http://www.ietf.org/rfc/rfc2608.txt
[9] UPnP Simple Service Discovery Protocol - http://upnp.org/sdcps-and-certification/standards/
[10] IETF Neighbour Discovery Protocol - http://tools.ietf.org/html/rfc4861
[11] IETF Domain Name Specification - http://www.ietf.org/rfc/rfc1035.txt
[12] Bluetooth Service Discovery Protocol

UDDI[13]) 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 pSHIELD 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.

The discovery engine developed for the pSHIELD purposes is depicted in Figure 4.2 and is composed by the following bundles:

- Discovery Engine Bundle: it is in charge to handle the queries coming from the *IGenericDiscovery()* interface. The Discovery Engine Bundle 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 Bundle using the *IServicesFilter()* interface to discard those components not matching with the enriched query.
- Query Pre-processor Bundle**:** 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 Bundle: 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 Bundle: 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.

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

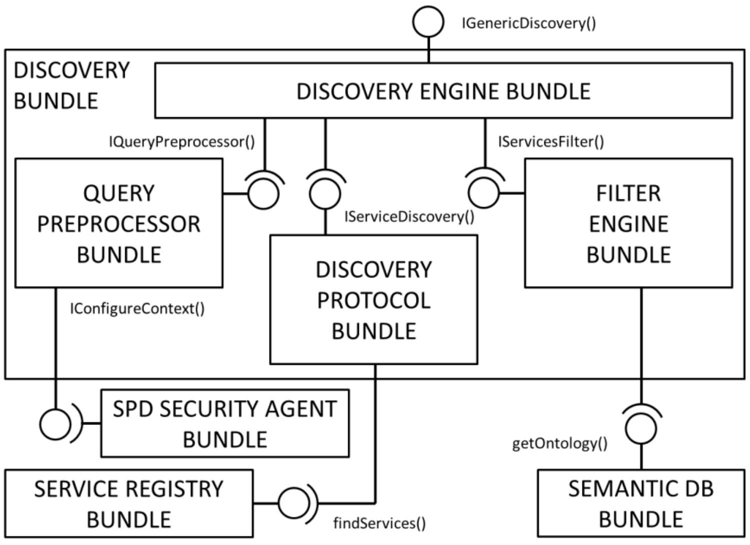


Figure 4.2: Discovery engine structure

The assessment of this technology is reported below:

Table 4.1: pSHIELD discovery engine – ASSESSMENT

pSHIELD discovery engine - ASSESSMENT	
Limits of this approach to be overcome	The discovery engine developed for pSHIELD application scenario is good to perform composability, but is not able to increase the SPD level of the system, since it introduces by itself a set of vulnerabilities. These vulnerabilities must be faced either with the adoption of “secure” discovery protocols, or with the introduction of additional functionalities.
Positive aspects to be preserved	The discovery engine is independent from any underlying discovery protocol, and this eases its implementation in a real environment. This feature must be preserved.

4.1.2 pSHIELD composition engine

Service composition 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 SHIELD Middleware Adapter functionality that cooperates with the SHIELD Overlay in order to apply the configuration strategy decided by the Control Algorithms residing in the SHIELD 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.

The discovery engine developed for the pSHIELD purposes is depicted in Figure 4.3 and is composed by the following bundles:

- <u>Composition Engine Bundle</u>: 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 Bundle use the *findServices()* interface to discover any suitable SPD component to be composed. Then the Composition Bundle 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 (that is the OSGi framework itself).

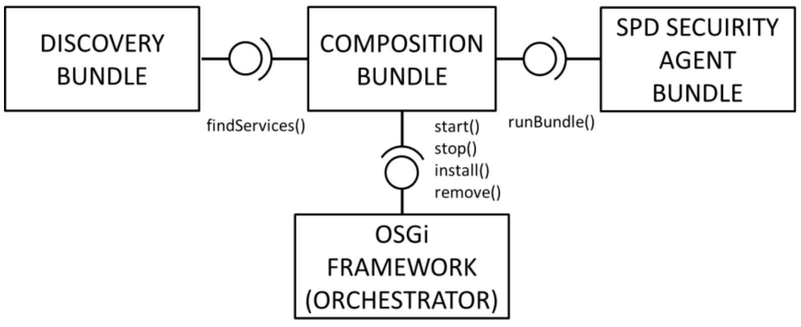


Figure 4.3: Composition Bundle

The assessment of this technology is reported below:

Table 4.2 pSHIELD composition engine – ASSESSMENT

pSHIELD composition engine - ASSESSMENT	
Limits of this approach to be overcome	The pSHIELD composition engine performs a valid composition of services, but as for the discovery engine, it increases the vulnerability of the system, in case an attacker forces the activation of malicious bundles. A mechanism to assure a trusted composition is needed.
Positive aspects to be preserved	The composition mechanism is decoupled from the composition orchestration, so it is reduced to the implementation of the decision taken from other modules. This eases the deployment, while preserving the modularity and scalability of the "intelligent part" that drive the composability (e.g. it works either with a choreographer or with an orchestrator).

4.1.3 pSHIELD orchestration engine

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

4.1.3.1 The OSGi framework

Considering the possible available SOA open solutions, our decision was to select OSGi as the reference service platform to develop the proof-of-concept demonstrator (and to realise the service orchestration). The main reasons leading to this decision were:

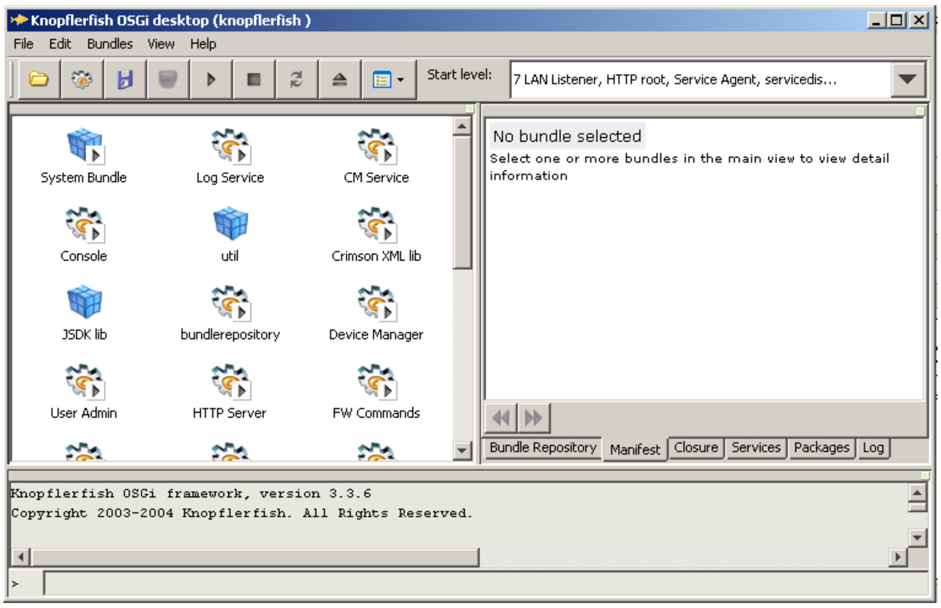
- OSGi is an open standard;
- OSGi has a number of open source implementation (Equinox, Oscar, Knopflerfish);

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

More in particular we decided to use the open source Knopflerfish OSGi service platform. Knopflerfish (hereafter referred as to KF) is a component-based framework for Java in which units of resources called bundles can be installed. Bundles can export services or run processes, and have their dependencies managed, such that a bundle can be expected to have its requirements managed by the container. Each bundle can also have its own internal classpath, so that it can serve as an independent unit, should that be desirable. All of this is standardized such that any valid Knopflerfish bundle can be installed in any valid OSGi container (Oscar, Equinox or any other).

Basically, running OSGi is very simple: one grabs one of the OSGi container implementations (Equinox, Felix, Knopflerfish, ProSyst, Oscar, etc.) and executes the container's boot process; much like one runs a Java EE server. Like Java EE, each container has a different start-up environment and slightly different capabilities. The KF environment can be downloaded here: http://www.knopflerfish.org/

The KF start-up environment is shown below in Figure 4.4.



```
Knopflerfish OSGi framework, version 3.3.6
Copyright 2003-2004 Knopflerfish. All Rights Reserved.
```

Figure 4.4: pSHIELD service orchestration engine: the Knopflerfish start-up environment

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

In conclusion, the assessment of this activity is reported below:

Table 4.3: pSHIELD orchestration engine – ASSESSMENT

pSHIELD orchestration engine - ASSESSMENT	
Limits of this approach to be overcome	The orchestration is a centralized approach that may introduce a single failure point, in case the orchestrator fails. This risk has to be mitigated either by physical redundancy or by the adoption of distributed approaches.
Positive aspects to be preserved	The OSGI Framework represents a solid base for the SHIELD Middleware implementation. The developed bundles, being written in Java language, are interoperable with other technologies by definition.

4.1.4 pSHIELD data and metadata management

The core services data management is an important issue, since the middleware should be provided with all the necessary information to activate services in a SPD/Context aware way. This has been achieved by decoupling the data management in two elements:

- *Service Registry Bundle*: it is in charge to store the bundle (i.e. SPD component) description in terms of provided functionalities, interfaces, semantic references, etc… Any pSHIELD Node, Network or Middleware layer component can be registered here to be discovered by its own proper pSHIELD Adapter. The Adapter registers each bundle as a service
- *Semantic DB Bundle*: it is in charge to store properly the semantic set by each Adapter Bundle. The stored ontologies contain all the information to compose the available Innovative SPD functionalities. The Semantic DB Bundle provides access to the SHIELD ontologies.

This is depicted in Figure 4.5

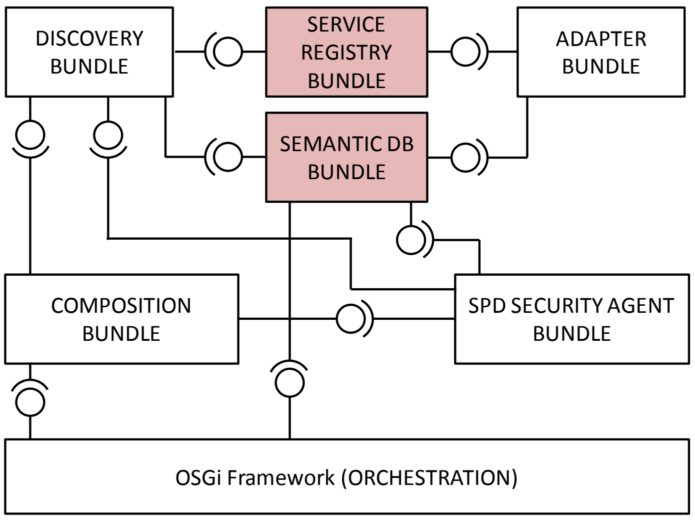


Figure 4.5: Middleware core service data management

The **service registry bundle** has been implemented with OWL-S. OWL-S, is the result of a collaborative effort by researchers at several universities and organizations, including BBN Technologies, Carnegie Mellon University, Nokia Research Centre, Stanford University, SRI International, USC Information Sciences Institute, University of Maryland, Baltimore County, University of Toronto, Vrije Universiteit Amsterdam, University of Southampton, De Montfort University and Yale University.

The first approach for Semantic Web services has been provided by OWL-S [99, 100]. Using OWL as the description language, OWL-S defines an upper ontology for semantically describing Web services that is comprised of three top-level elements: A service in OWL-S is described by means of three elements, as shown in Figure 4.6.

1. The *Service Profile* describes what the service does. It explains what the service accomplishes, details limitations on its applicability and quality of service, and specifies requirements the service requester must satisfy to use it successfully. This information is used by consumers during the discovery of the service.
2. The *Service Process Model* describes how to use the service. It details the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step-by-step processes leading to those outcomes.
3. The *Service Grounding* specifies the details of how to access/invoke a service. It includes communication protocol, message formats, serialization techniques and transformations for each input and output, and other service-specific details such as port numbers used in contacting the service [103].

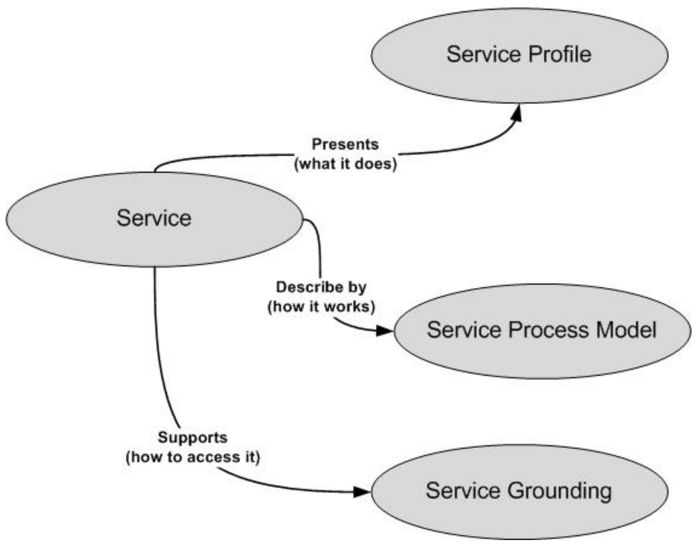


Figure 4.6: OWL-S Service Description Elements

Therewith, OWL-S provides a model for semantically describing Web services and serves as a basis for various research and development activities on Semantic Web service technologies [101]. However, OWL-S is criticized for conceptual weaknesses and incompleteness: the meaning of the description elements is not clearly defined and thus used ambiguously, leading to misinterpretations and incompatible service descriptions; furthermore, although OWL-S allows other languages like KIF and SWRL for process descriptions besides OWL, their formal intersection is not defined, hence a coherent formalism for semantically describing Web services is not provided [102]. However, the OWL-S provides developers with a strong language to describe the properties and capabilities of Web Services in such a way that the descriptions can be interpreted by a computer system in an automated manner [103]. In the following, we briefly summarize the underlying standard ontology language OWL and then present each of the main elements of OWL-S service descriptions.

The information provided by an OWL-S description includes

- ontological description of the inputs required by the service
- outputs that the service provides
- preconditions and post conditions of each invocation

The goal of OWL-S is to enable applications to discover, compose, and invoke Web Services dynamically. Dynamic service discovery, composition, and invocation will allow services to be introduced and removed seamlessly from a services-rich environment, without the need to modify application code. If the information it needs to achieve its goals can be described in terms of an ontology that is shared with service providers, an application will be able to detect new services automatically as they are introduced and adapt transparently as the programmatic interfaces of services change. Although it is not the only technology being pursued to support dynamic environments, OWL-S is far enough along in its development to be used as a proof of concept, if not a potential solution.

Consequently, the emerging concept of Semantic Web services aims at providing more sophisticated Web Service technologies along with support for the Semantic Web. Mentioned first in [104] and [105], Semantic Web services shall utilize ontology's as the underlying data model in order to support semantic interoperability between Web services and its clients and apply semantically enabled mechanisms for automated discovery, composition, conversation, and execution of Web Services. Therefore, exhaustive description frameworks are required that define the semantic annotations of Web services needed for automatically determine their usability.

The **semantic DB bundle** has been simply populated with the ontology developed for pSHIELD purposes (see previous section).

The assessment of this technology is reported below:

Table 4.4: pSHIELD data and metadata management – ASSESSMENT

pSHIELD data and metadata management - ASSESSMENT	
Limits of this approach to be overcome	The data management mechanism adopted for the pilot phase worked very well and was able to produce solutions as expected. However, in order to enrich the potentiality of the SHIELD composability, it could be useful to enrich the information stored or exchanged at middleware level (this is strictly related to the development of the new SHIELD semantic models).
Positive aspects to be preserved	The decoupling architecture for data management should be preserved, as well as the use of a standard for service description (OWL-S)

4.2 nSHIELD potential investigations

4.2.1 SHIELD secure service discovery and delivery

[…] This task will design and/or develop the core SPD services provided by the SHIELD middleware:

• service discovery entailing mechanisms to securely register, advertise, discover, locate, filter, rank and select the

In distributed networks it is imperative to have services discovered, composed and delivered to legitimate service users in a secure way. The same principle applies to embedded systems as well, with the added complexities that are imposed by their heterogeneous and resource-constrained nature.

4.2.1.1 The OASIS Standards

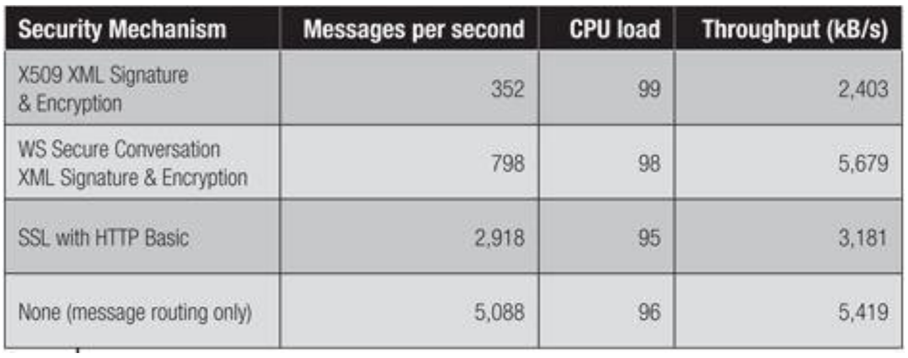
The Organization of Structured Information Standards (OASIS, [1]) has focused on standardizing web services provision and has gradually released a number of related standards. Many of these enjoy wide support from industry leaders and are already used in various applications, including current versions of software by Microsoft, Sun Microsystems, Oracle, Apache, RSA Security Inc., Verisign etc.

4.2.1.1.1 WS-Security

The Web Services Security Specification (WS-Security or WSS, [2]) is part of the WS-* family of specifications published by OASIS. It was originally developed by IBM, Microsoft and Verisign and was presented on April 2002 as an extension to SOAP aiming at providing end-to-end security to web services (mainly via the use of XML Digital Signature and XML Encryption). The current version is 1.1 which was released on February 2006.

The protocol specifies enhancements to existing SOAP messaging, integrating security features in the header of SOAP messages (working in the application layer), in order to provide additional security-related functionality such as message-level confidentiality, integrity and authentication to SOAP messages. The main mechanisms detail signing SOAP messages (integrity, non-repudiation), encrypting SOAP messages (confidentiality) and attaching security tokens to SOAP messages (authentication). There is a variety of supported encryption, signature and security token formats. The latter include SAML Assertions [3], Kerberos tickets [4], X.509 Certificates [5], Rights Expression Language (REL) Tokens [6], UserID/Password credentials [7] as well as custom tokens.

It should be noted that the aforementioned WSS mechanisms should not be considered a comprehensive security solution for Web services, but merely a building block to be using in conjunction with other protocols and web service extensions. Moreover in situations where point-to-point confidentiality and integrity are adequate, Transport Layer Security (TLS) could be considered an alternative. Unlike WSS though, TLS cannot offer end-to-end (message-level) security and it is not as flexible as when application-level proxy servers are involved. Still, the performance overhead is significant with the standard WS-Security implementation (see Figure 4.7) and this is an area where further research is required (and already being conducted) in order to improve its usability in resource constrained devices.



Security Mechanism	Messages per second	CPU load	Throughput (kB/s)
X509 XML Signature & Encryption	352	99	2,403
WS Secure Conversation XML Signature & Encryption	798	98	5,679
SSL with HTTP Basic	2,918	95	3,181
None (message routing only)	5,088	96	5,419

Figure 4.7: Benchmark Results for 25 concurrent requestors, [8].

4.2.1.1.2 WS-Trust

Web Services Trust Language (WS-Trust [9], approved March 2007, at v1.4 since February 2009) is an important add-on to WS-Security, defining primitives that allow the issuance, exchange, renewal and validation of security tokens and thus the establishment and assessment of trust relationship, even between entities across different trust domains. It introduces a Security Token Service (STS) which is responsible for issuing WS-Security tokens, defines the format of security token requests and their responses as well as mechanisms for key exchange.

4.2.1.1.3 WS-SecureConversation

Web Services Secure Conversation (WS-SecureConversation v1.4, [10]) is another WS-Security add-on this introduces, similarly to TLS, a session key to secure communication across one or more messages. The aim of the specification is to establish security context, share, renew, amend or cancel said context as well as derive (potentially more efficient) sessions keys from the abovementioned context.

When multiple message exchanges are involved WS-SecurityConversation has proven to be more efficient than a plain WS-Security implementation (e.g. [11]) but the former requires the presence of other WS-* protocols as well, like WS-Trust, so the added complexity should also be considered.

4.2.1.1.4 WS-SecurityPolicy

Web Services Security Policy (WS-SecurityPolicy, at v1.3 as of February 2009, [12]) defines a set of security policies (to be used along with the Web Services Policy Framework, endorsed by W3C [13]) enabling web services to define security-related constraints and requirements regarding. These policies extend the functionality and flexibility of the protocols defined in WS-Security, WS-Trust and WS-SecureConversion, improving compatibility and interoperability of web services and participants.

Policy assertions include protection assertions (identifying parts of a message that require confidentiality, integrity checks etc.), token assertions (specifying which token formats are allowed), security binding assertions (defining cryptographic algorithm suites, transport layer security, timestamps etc.) and supporting token assertions (adding supporting function like username/pass user sign-on).

4.2.1.1.5 WS-Discovery

The OASIS specification Web Services Dynamic Discovery v1.1 (WS-Discovery, [14]) approved in 2009 as an OASIS standard, defines a service-discovery protocol, operating either in ad-hoc or managed mode. In ad-hoc mode probes are used (sent to a multicast group) and matching services send a response directly to the requester. The managed mode of operation, which is more scalable, involves a discovery proxy to which services send announcements when joining and leaving the network.

4.2.1.1.6 WS-Federation

Web Services Federation Language (WS-Federation , at v1.2 as of May 2009) details mechanisms which extend the functionality of the WS-* protocols to allow communication between different security realms, enabling authorized access to resources belonging in one realm to entities whose identities and security attributes are define in another realm.

4.2.1.2 Devices Profile for Web Services - DPWS

The communication and service composition in complex architecture where various types of devices requires interacting with each other it's not trivial problem. This is because most digital devices (especially those with embedded systems) are equipped with their own architectural components. In such cases proxies can be utilized to overcome incompatibility issues. However, this approach can have various limitations e.g. (limited number of support devices, high cost, etc.). Thus, application of middleware concepts that are widespread in other application domains and may be used to solve interoperability problems.

Web services provide an ideal framework to address issues such as heterogeneity and interoperability. However, it is of major importance to understand the performance limits and constraints, in terms of resource requirements, imposed by various Web services toolkits, in order to estimate expected performance at run-time (i.e., executing a task at hand).

The Devices Profile for Web Services (DPWS) [40] is such a cross domain technology for inter machine communication and base on Web Services. DPWS employs similar messaging mechanisms as the Web Services Architecture (WSA) [41], with restrictions to complexity and message size. The DPWS defines a minimal set of implementation constraints to enable secure Web Service messaging, discovery, description, and eventing on resource-constrained devices. It features secure message exchange, dynamic discovery, and description of devices based on WS-Discovery, WSMetadataExchange and WS-Transfer. Furthermore, it provides a publish-subscribe eventing mechanisms based on WS-Eventing. It should be noted that DPWS requires nodes to have an IP connectivity, otherwise, is not able to utilize such DPWS feature. This can be achieved using the 6LoWPAN [42].

The DPWS offer the ability to resource constraint devices (e.g. limited memory and battery energy), to provide services similar to those offered by traditional web services. Currently various Application Programming Interfaces (API) can be used to incorporate DPWS in embedded device. Toolkits in C++ as well as Java/J2ME (e.g., Symbian based devices), and furthermore .NET implementations on Microsoft platforms are available for developing web services. In particular:

C/C++ toolkits: WS4D-Gsoap [43] is a toolkit for developing Web services consumer and providers being conform to the "Devices Profile for Web Services" (DPWS). The toolkit is implemented as an extension to the well-known gSOAP Web services toolkit. This toolkit offers multi-platform support such as Linux i386, Windows-native, Windows-cygwin and embedded Linux.

The **.NET Compact Framework** (CF) is a subset of Microsoft's .NET framework. The .NET CF is supported on various devices/platforms that are based on PocketPC and Smartphone architectures. Web services on .NET CF support the use of synchronous or asynchronous invocation [17]. Development of embedded Web services is analog to implementing Web services clients in .NE.

The **Java 2 Platform Micro Edition** (J2ME) is a set of standard Java APIs defined through the Java Community Process (JCP). The J2ME specifications define the Connected Device Configuration (CDC) (i.e., a subset of J2SE) and the Connected Limited Device Configuration (CLDC). In contrast to CDC, CLDC provides libraries such as the Connection Framework which are suitable for devices with a small memory footprint (not part of J2SE). CLDC targets hardware platforms with 128 KB to 512 KB memory and 16-bit or 32-bit CPUs. Different SOAP APIs and Web services toolkits are suitable for J2ME/MIDP based devices. Such examples are the following:

• DPWS4J	(https://forge.soa4d.org/projects/dpws4j/)
• kSOAP 2	(http://ksoap2.sourceforge.net/)
• JMEDS	(http://sourceforge.net/projects/ws4d-javame/)

The choice of a particular DPWS technology depends on the type of service that will be used in nSHIELD.

Considering the dynamic multilayer architecture of nSHIELD (see Figure 4.8) the different elements/components should be able to discover, register in and integrate different services when is needed securely.

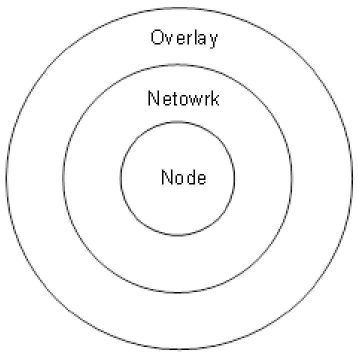


Figure 4.8: nSHIELD abstract architecture

All the components in the nSHIELD architecture should be able to define their authentication mechanism as well as enable access control services to restrict the access in information only to those components that need access to a particular set of data.

For instance, consider a case where an overlay service requires information to monitor the state of underlying nodes. If we assume nodes are connected dynamically to nSHIELD infrastructure the overlay is not possible to know all the nodes in advance. In that case the overlay should multicast a discovery message on the managed domain to find the nodes that should receive information. Alternatively, if there is a discovery service directory the overlay can request the registered services by the discovery service. The discovery service typically stores the network location information of services that are present on the local subnet as well as on a wider network and allows for discovery of such services. This way we can decrease the network traffic compared to the multicast discovery. The overlay to identify the discovery service can send a multicast WS-Discovery *probe* message explicitly looking for a discovery proxy on the network or can be pre-configured with the discovery service information.

Considering that this discovery service is available every component participating in the SHIELD architecture should introduce them to the discovery service. The joined component announces its features to the discovery service via WS-Discovery hello message. The discover service the hello message and stores the information about the services of the joined component (e.g motion detection). The overlay requires access to the information provided by the joined component sends WS-Discovery probe message to the discovery service searching for motion detection. The discovery service responds with a WS-Discovery probe match message carrying the network location information of the motion detection service. Since the overlay receive this information is able to connect and use the provided service. This procedure is illustrated in Figure 4.9.

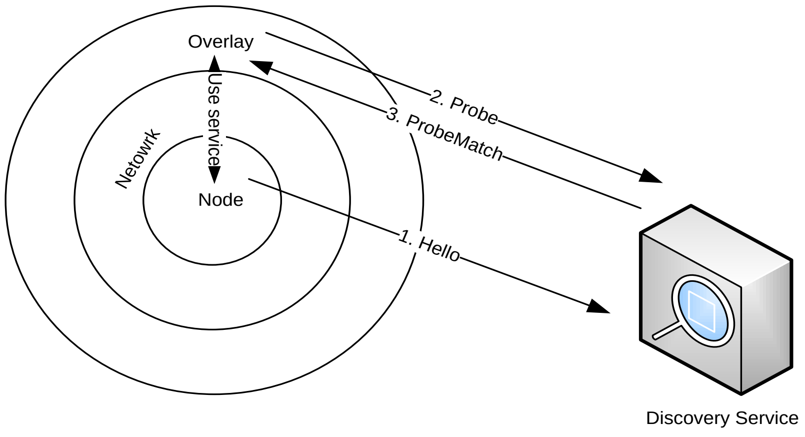


Figure 4.9: An example of discovering service in the nSHIELD architecture

Though we believe that DPWS is the most suitable technology to integrate nSHIELD different components on a unified architecture, requires more research work to address possible issues arising from the particular architecture of nSHIELD. For instance, which are the "consequences" of employing the DPWS in micro nodes of nSHIELD? Do all the available APIs have the same consequences? How the defined scenarios can affect the choice of using DPWS?

4.2.1.2.1 Security for Devices Profile for Web Services

The DPWS specification provides various mechanisms to enable security, depending on deployment requirements. The transaction among a client and a device may require authentication, integrity and confidentiality services.

The DPWS propose the utilization of the following mechanisms:

Transport Layer Security (TLS): TLS [59] is used to establish an end-to-end secure channel between the communicating entities. It does not only provide integrity, authenticity and confidentiality but also identity verification if exploits X.509 certificates.

HTTP Authentication: HTTP Authentication [60] is used to authenticate users, where the client provides user name and password. However, if the basic http authentication scheme is employ we highly recommend using it after TLS session establishment.

WS-Discovery Compact Signatures: The WS-Discovery [61] relies on multicast and unicast UDP datagrams for transport. However, TLS cannot be utilized to protect connectionless protocols such as UDP. Thus, the DPWS proposes the WS-Discovery Compact Signatures which allows a client to verify the integrity of discovery messages, and to identify WS-Discovery traffic that was signed by a device with a specific cryptographic credential. This type of compact signatures is based on XML signatures [62].

In the case of nSHIELD as a *client* can be any component of the upper networking layers.

Threats: Threats and attacks that nSHIELD should deal with is not different from those facing traditional wired and wireless Internet based services. Vulnerabilities can be also identified in the software stack. For instance, in [63] is published a vulnerability to the DPWS Microsoft's API that could allow remote code execution. The following table summarizes the possible vulnerabilities.

Table 4.5: nSHIELD possible threats that can face Micronodes incorporating DPWS

Threat/Attack	Brief Description	Consequences
Impersonation	An attacker may impersonate a micronode in order to send modified data to the upper layers.	Unauthorized access, integrity loss
Node Integrity	An attacker may modify node's configuration data in order to either get unauthorized access in the micronode.	Denial of service, unauthorized access
Confidentiality loss	An attacker may try to break the secure channels to gain unauthorized access to the data provided to the upper layers.	Unauthorized access
Availability loss	An attacker may try to cause micronodes services unavailable.	Denial of service
Software vulnerabilities	An attacker may exploit a published vulnerability in order to gain either unauthorized access or to cause a denial of service	Unauthorized access, Denial of service

4.2.1.3 DPWS – Implementing the OASIS Standards on Resource-Constrained Devices

The need for implementing dynamic and secure discovery of devices and Web Services (including messaging, description, interactions, event-driven changed etc.) on resource constrained devices led to the development of the Devices Profile for Web Services specification (DPWS, [16], at v1.1 since July 2009).

The profile's architecture includes hosting and hosted services, where the former are associated to a device and are essential for device discovery and the latter are functional and reply on the hosting device for discovery. Figure 4.7 provides an overview of the client, device and service arrangement.

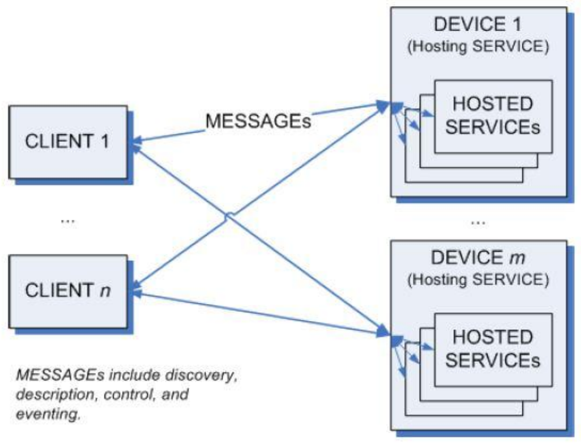


Figure 4.10: Arrangement of clients and devices [16]

Moreover, discovery services are included, enabling devices to "advertise" their presence on the network and search for other devices. Metadata exchange services provide dynamic access to services hosted on a device and their metadata and publish/subscribe eventing services allow other devices to subscribe to messages provided by a certain service. The DPWS protocol stack can be seen in Figure 4.11.

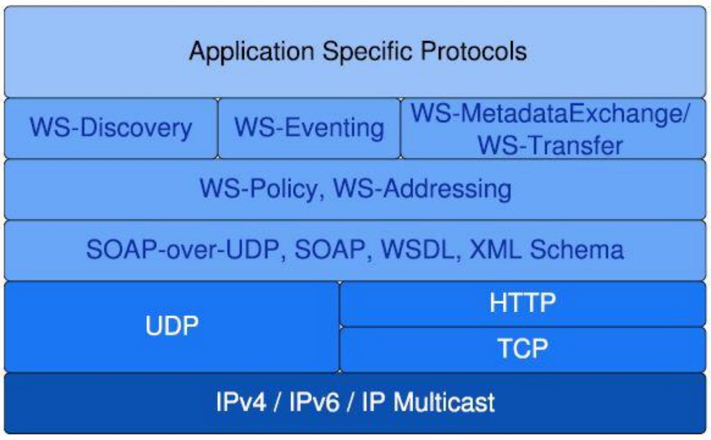

Application Specific Protocols
WS-Discovery
WS-Eventing
WS-MetadataExchange/ WS-Transfer
WS-Policy, WS-Addressing
SOAP-over-UDP, SOAP, WSDL, XML Schema
UDP
HTTP
TCP
IPv4 / IPv6 / IP Multicast

Figure 4.11: The Devices Profile for Web Services Protocol stack [16].

The EU research project "Service Infrastructure for Real time Embedded Networked Applications" (SIRENA, [18]) was one of the earliest implementation of the DPWS on embedded devices. Their results were a foundation for the EU projects "Service-Oriented Device & Delivery Architecture" (SODA, [19]) and "Service-Oriented Cross-layer Infrastructure for Distributed Smart Embedded Devices" (SOCRADES, [20]) that followed but also led to the introduction of the Service-Oriented Architecture for Devices (SOA4D, [21]) and Web Services for Devices (WS4D, [22]) open source programs.

4.2.1.4 Service-Oriented Architecture for Devices - SOA4D

SOA4D provides development toolkits (in C and Java, simplifying the development of DPWS-compliant applications (and thus the implementation of the WS-* family of protocols) for embedded devices. Figure 4.12 provides a graphical representation of the SOA4D architecture.

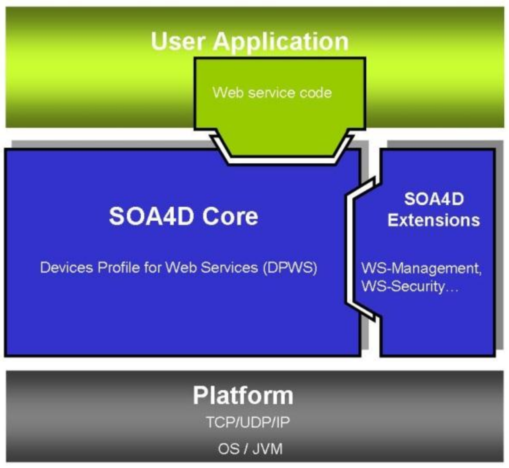


Figure 4.12: The SOA4D Architecture [21]

4.2.1.4.1 Web Services for Devices – WS4D

WS4D is an open source initiative which provides a number of toolkits aimed at developing DPWS-compliant applications for resource-constrained devices in ad-hoc networks which are interoperable with regular W3C-specified Web Services.

Toolkits include the WS4D-gSOAP (C/C++ languages) and WS4D-JMEDS (Java) as well as the WS4D-Axis2, a Java-based Apache project. WS4D also includes two toolkits which utilize state of the art technologies, namely WS4D-jCoAP and WS4D-uDPWS. WS4D-jCoAP is a Java-based implementation using the Constrained Application Protocol (CoAP, [23]), a protocol currently in IETF draft form which is promoted as an alternative to HTTP optimized for highly resource-constrained devices and networks. WS4D-uDPWS is a proof-of-concept implementation of DPWS for the emerging IETF IPv6 over Low power Wireless Personal Area Networks (6LoWPAN, [24]) protocols.

A detailed overview of the WS4D initiative can be found in [17].

4.2.1.4.2 Other Implementations

Other than the EU projects already mentioned, “Network-centric Middleware for group communications and resource sharing across heterogeneous embedded systems” (MORE,[25]) also focused on the development of a DPWS-compatible middleware. In fact the project’s Core Management Service was based on an implementation of DPWS4J, the open-source Java-based version available from the SOA4D project, for embedded devices. A description of a Service Orchestration mechanism aimed at resource-constrained devices deployed in hierarchical network topologies and which is to be applied to services running on top of MORE’s DPWS-based middleware can be found in [26]. In [27] a combination of the DPWS stack with a low footprint P2P network technology is presented, including real-life measurements which show that this approach outperforms SOAP-based methods when communications take place via narrow-bandwidth cellular networks.

4.2.1.5 Other State-of-the-Art Secure Service Discovery and Delivery Technologies

Other than the OASIS standards, there are various approaches and protocols pertaining to secure service discovery, composition and delivery for embedded devices.

A thorough survey of Service Composition technologies in ambient intelligence environments can be found in [28]. Taxonomy of security-aware service composition approaches is presented in [29], along with a comparative evaluation of identified technologies. The classification includes Security-aware Syntactic-based Approaches (Information Flow Control based, Access Control Based) and Security-Aware Semantic-based Approaches. Another relevant survey can be found in [30], where the focus is on service composition middleware and which are classified based on a four step model (translation, generation, evaluation, execution) and pervasive requirements (interoperability, discoverability, adaptability, context awareness, QoS management, security, spontaneous management and autonomous management).

Minimizing service disruptions is critical, especially for ad-hoc applications, and [31] presents a service composition and recovery framework aiming to address that, via service routing (selecting the service components that support the service path) and network routing (finding the optimal network path that connects the service components). A disruption index is also introduces, aiming to characterize aspects that are undetectable by other metrics (e.g. reliability and availability). ReSCo is a relevant approach, a lightweight middleware for reliable service composition in pervasive dynamic systems, which is presented in [32], along with simulations that demonstrate the effectiveness of the proposed mechanisms. Furthermore, [33] proposes the design and development of an intelligent service management component (with two different proposed methods – one based on Sensor-Service Ontology reasoning and the other on Rule-based reasoning) to be included in the middleware layer of ubiquitous sensor networks.

Secure service discovery is another critical area of research. A secure service discovery model for pervasive environments is presented in [34], using a hybrid secure and non-secure discovery scheme and based on mutual trust. [35] proposes a service discovery scheme based on stable and resource-rich nodes identified as “volunteers” and additional trust management mechanisms. A similar model is introduced in [36], classifying services into three levels, with different levels being discovered in different manners, while [37] and [38] focus on ad-hoc networks with the former proposing a hybrid peer-to-peer/directory-based scheme and the latter a Secure Pervasive Discovery Protocol (SPDP) based on an anarchy trust model. Finally, a decentralized approach based on trust management principles is used in the Flexible and Secure Service Discovery (FSSD) protocol introduced in [39].

4.2.2 SHIELD trusted service composition

[…] • service composition entailing mechanisms to automatically resolve the dependencies and to discover, deliver and deploy the atomic services; […]

4.2.2.1 Web Service Composition

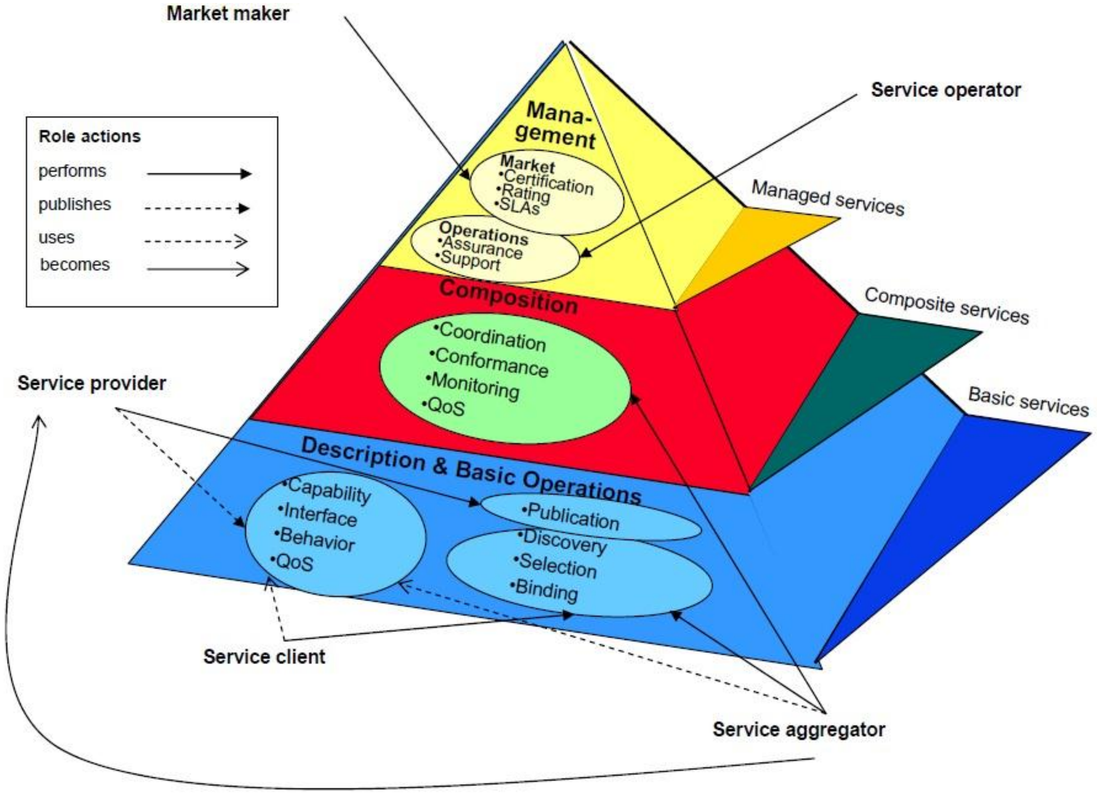


Figure 4.13 Service-Oriented Computing (SOC) Pyramid

Perform Composition

Web service composition tries to effectively and efficiently compose existing services with minimum user intervention, in order to address needs that cannot be satisfied by an atomic service. It aims to combine the functionalities offered by different services to develop a value-added composition service. Also, the definition of increasingly complex application is allowed, by progressively combining services at increasing levels of abstraction. Service composition accelerates rapid application development, service reuse and complex service consummation.

Dynamic composition

The proposed composition approaches can differ on *'when'* and *'how'* the composition schema is created. The term '*when'* is referred to static or design-time composition and dynamic or run-time composition. In the first case services are chosen, linked and compiled before runtime and the schema does not change. In the second case the composition schema is adapted to reflect unpredictable changes and effects at runtime. The term 'how' refers to manual and automated composition functionalities.

There are 4 service composition models:

- Orchestration
- Choreography
- Coordination
- Service Component Architecture

Orchestration, which was adapted for the pSHIELD core services composition process, is a centrally controlled decision making specific business process that is executed by a single entity. It organizes the participating services into a process flow and describes the interaction between them as well as the control and data flow. Furthermore, the services can be orchestrated recursively. WS-BPEL is an OASIS standard and has been generally adopted as a standard orchestration language.

On the other hand **choreography** is a multi-party collaboration for distributed decision making as part of some larger business transaction. It describes the collaborations between the multiple partners and focuses only on the conversational aspect of the interaction. It does not aim to expose what's underneath, like the internal processes of the partners, and it is not directly executable. Thus, Choreography can be combined with orchestration. Orchestration is used for the description of the internal processes, while choreography is used for the description of the global external interaction between the participants. WS-CDL and WSCI are W3C standards and the most relevant standard languages for choreography modelling.

In **Coordination**, a group of participating services interacts, following a coordination protocol. In that case, a coordinator is needed, as a special entity, to enforce the coordination protocol and decide on the outcome of the interaction after it is finished. The participants are communicating through the coordinator. They don't have to communicate with each other as in choreography. WS-Coordination and WS-CF are used to model coordination processes.

In **Service Component Architecture**, components are configured instances of service implementations which provide business functions and can have settable properties. The business functions are offered of use by other components as services. The implementations may depend on services that are provided by other components. Finally, the composition can contain all the above plus the wiring that describes the connections between the elements. This model was created by the OSOA Consortium.

Table 4.6: Service Composition in the Semantic Web

SWS Languages	SWS Discovery Support	Process Model	Inter-Process Communication
OWL-S	Yes	Limited Orchestration	Synchronous
WSMO	Yes	Orchestration/Choreography	Synchronous/Asynchronous
BPEL4SWS	Through WSMO	Orchestration/Choreography	Synchronous/Asynchronous

4.2.2.2 Automated Service Composition

Automated service composition attempts to decrease the human intervention in the composition process.

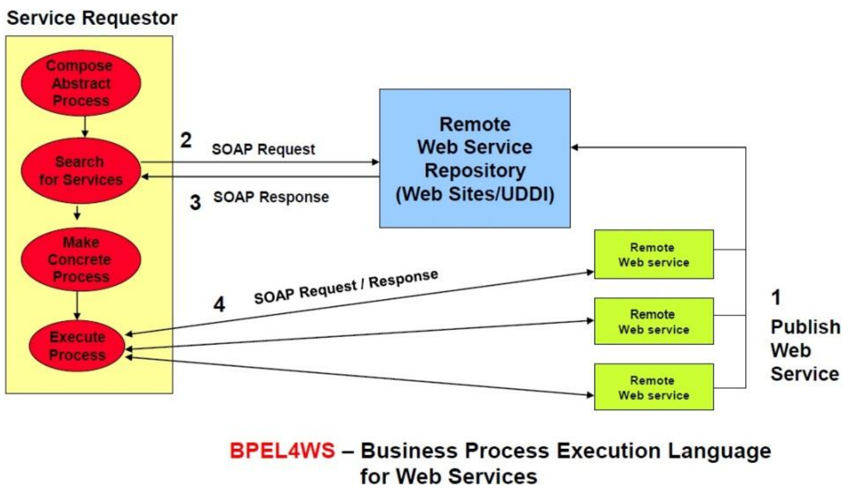


Figure 4.14: Service composition

Automatic composition involves five phases:

1. **Advertisement:** the service providers advertise their services that will be used in the composition, usually with semantic descriptions

2. **Translation:** the external descriptions are then translated to internal ones that describe the processes with semantic descriptions
3. **Process model:** it is created combining the existing service descriptions to satisfy a user request
4. **Evaluation:** the resulting process models are evaluated based on non-functional attributes
5. **Execution:** one process model is chosen from the previous phase and is executed

There are two main approaches for implementing automated service composition:

- **Workflow techniques:** they view services as flows and exploit the accumulated knowledge of the workflow community and apply it in the service domain. They automatically generate an abstract workflow based on a high-level goal. Then, a reasoning service aims to decompose the goal and tries to satisfy it with a composition of tasks. The abstract workflow tasks are matched with concrete services or compositions of those. The result of these techniques is a concrete workflow which is executed.
- **AI planning techniques:** they generate a plan that contains the series of actions required to reach the goal state set that has been set by the service requester, beginning from an initial state. The planning problem is expressed using a domain theory like classical logic, graphs, hierarchical task networks and finite state machines.

Other approaches include:

- **Model-driven service composition:** in this approach, concepts from Model-driven development in software engineering are used. Here, all software functionality is specified using platform independent models (PIMs). Then, PIMs are translated to various platform-specific models (PSMs). PSMs can either be compiled into executable code or run directly. In a proposed instantiation, BPMN was used as a PIM and BPEL as a PSM.
- **QoS-aware Service Composition:** except from functional attributes, QoS attributes are taken into consideration to augment service selection. There are two phases involved to the QoS calculation:
    1. The **Local optimization:** in this phase, QoS is calculated for each service individually
    2. The **Global optimization:** here a globally optimum set of services is found instead of a set of individually optimum services, in order to satisfy global QoS constraints

QoS approach takes into consideration attributes for performance, like time calculations, throughput and scalability, and for dependability, such as availability, accuracy and robustness.

In [52], the authors proposed a novel approach to automated composition of services based on their security policies. Given a community of services and a goal service, they reduce the problem of composing the goal from services in the community to a security problem where an intruder should intercept and redirect messages from the service community and a client service till reaching a satisfying state. The implement the algorithm in AVANTSSAR (Automated VAlidatioN of Trust and Security of Service-oriented ARchitectures) Platform and applied the tool to several case studies.

4.2.2.3 Trusted service composition

The service composition is a challenging task that involves the integration of many existing services to implement more complex processes. 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. A good survey on trust in semantic web services can be found in [44].

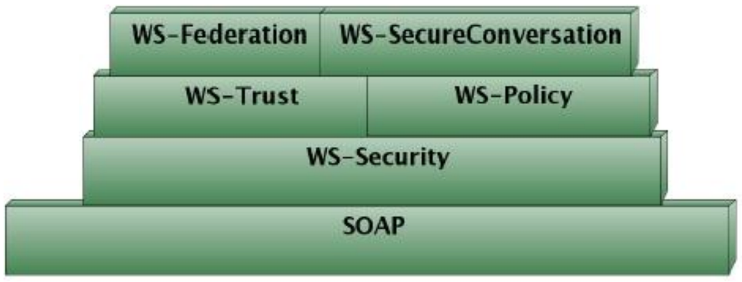


Figure 4.15: Web Service Security Stack

There are two main types of trust. The first one is called *soft trust* or *reputation based trust*. It is community dependent and participants exchange information about the other known participants and services. Malicious entities can be detected and isolated based on the exchanged information. The drawback is that no one takes the risk to invoke a new unknown service initially before deciding the trustworthiness of the new services for invocation. The second type is called *hard trust* or *certificated based trust*. Here trust does not depend on the social context. The semantics of security behaviour can be specified in service description. The trustworthiness can be derived from the policy or contract. The drawback is that anyone can provide a fake or wrong policy or contract.

Trust is also classified according to the origin of the trust information. **Direct trust** is the opinion that an entity possesses about other entities and is determined by their previous interactions. **Indirect trust** stands for the opinion that other entities possess about the investigated entity. Usually an entity consults the indirect trust of other entities that are considered trustworthy. Indirect trust is further categorized in reputation, recommendations and referrals. Reputation is the general feedback about an entity's behaviour. Recommendation means that a user trusts a service because of some suggestion got from a trusted central authority. Referral is that a service consumer trusts a service because of some distributed referrals got from known trusted third party agents.

Based on the topology and the way that trust information is established three models are categorized:

1. **Peer-to-Peer trust model:** the model is not scalable, since the participants should share their trust relations on one-by-one basis
2. **Trust Chain Model (TC):** each participant stabilizes its trust relation with its direct ancestor, which is not flexible for dynamic service integration
3. **Trusted Third Party Model (TTP):** here all the participants get the credentials from a trusted third party

In [45] the authors presented a trust and reputation model for web service selection based on Bayesian networks. They consider three sources as trust information: useful reputation, QoS monitoring and direct experience of consumer. The model tries to overcome some limitations in trust calculation by integrating the mentioned sources to find the final trust value.

In [49], an approach is proposed to deal with malicious entities that try to gain high trust value in order to perform malicious activities later. A penalty vector is introduced, which represents the misbehaviour in past transactions which has been derived from inconsistency and misuse of trust measure by peers. The malicious service providers can be found by using the result of subtraction of penalty vector from trust vector.

In [46] the authors follow the Reputation based and Trust Third Party model. They propose a Priority based trust (PB) model for service selection in general service oriented environments, which overcomes the limitations of Certified Reputation Model.

In [47] the authors focus in developing a trust model according to the behaviours of consumers and providers at dynamic environments. Instead of only concentrating their attention only to provider behaviour, they also consider the reputation of the consumer. Consumer reputation is collected by other consumers in the same community and by its interaction with other communities.

The authors in [48] try to model the behaviour of a service. The trustworthiness is categorized as objective and subjective. The objective features are customer requirement, capability of implementing functionalities, commitment to complete work, performance and existence. The features subjective trustworthiness is reliability, honesty and expectation. They also use the OWL-S to structure the ontology schemas.

In [50], a framework for automatic selection and composition of services is presented, which exploits trustworthiness of services as a metric for measuring the quality of service composition. Trustworthiness is defined in terms of service reputation extracted from user profiles. The profiles are extracted and inferred from a social network which accumulates users past experience with corresponding services. Using the proposed privacy inference model, they prune social network to hide privacy sensitive contents and utilize a trust inference based algorithm to measure reputation score of each individual service and subsequently trustworthiness of the composition.

QoS-driven Trusted Composition Evaluation Model (TCEM) [51] can judge the business stream effectively and comprehensively by choosing an appropriate and trusted evaluation method for the generated services composition chains. Furthermore, on the basis of Case-Based Reasoning (CBR) and the retrieval mechanism oriented to web services coordination, the TCEM0based execution engine and algorithm is implemented to evaluate the quality of service composition. The proposed algorithm has the satisfied result compared with the traditional method and is shows more efficient and trustworthy where web services composition is widely used in multi domains.

4.2.2.4 LAs (Level Agreements) contributing to trusted composition

Although current LA languages allow expressing Quality of Service (QoS) constraints with different success, the absence of security and dependability aspects in LAs makes it difficult to create agile QoS for Embedded systems.

A Service Level Agreement (SLA) (kind of LA) is a common way to specify the conditions under which a service is to be delivered, but is usually limited to availability guarantees.

The most well-known machine-readable SLA models are the Open Grid Forum’s Web Services Agreement (WS-Agreement) [53] and IBM’s Web Service Level Agreement (WSLA) [54]. The WS-Agreement specification proposes a domain-independent and standard way to create SLAs and has been up taken in several projects, while its predecessor WSLA seems to be deprecated.

It is difficult to express that a SLA can be derived to establish a kind of level agreement among different embedded systems (at node layer). There is always has to be a control systems responsible for managing the interactions between nodes and this means software processing. However, nSHIELD could analyse the process of including agile Level Agreements in networks of embedded systems. This is one main issue that will be analysed within the project jointly with the analysis of metrics in WP2 as parameters for measuring and composing techniques for increasing QoS.

SLAs can serve as a contract for composing secure and dependable systems and have a fundamental role in QoS. Some challenges have to be solved around this field: what kind of contract we define between two entities and how this contact evolves in terms of security when there are more than 2 entities trying to communicate each other or consume one or more resources at the same time.

An example of trustworthy composition could be achieved by a mixture of SLAs and copotiion techniques. By monitoring agreement degradations (e.g., via SLAs) combining the information of current QoS measurement in the SLAs, the diagnosis decides whether to initiate recovery. The main recovery method of this approach is to instantiate a new service of the affected service’s type. Depending on performance degradation or service malfunction the new service rule is additional service or replacement. In both cases, the interceptor reroutes the request to the new service.

4.2.3 SHIELD service orchestration and choreography

[…] • service orchestration entailing run-time mechanisms to install, start, pause, stop, refresh and uninstall the services in a distributed environment. […]

Two mechanisms will be examined to manage (install, start, pause, stop, refresh uninstall) services in the SHIELD framework: the orchestration and the choreography.

4.2.3.1 OSGI framework for service orchestration

[…] A prototype of the proposed architecture will be implemented, a key feature of which will be the exploitation of ontology-based, semantic technologies to represent the SPD services model. Since an integrated support to SPD functionalities in middleware architectures is still largely unexplored in both academic and industrial research activities, this task will investigate how to extend the firstly emerging models to accomplish with the SHIELD requirements provided by WP2. Indeed the design and development of the SPD Middleware core SPD services will be accomplished according to the specifications, requirements and architectural guidelines coming from tasks 2.1 and 2.3. […]

Modularity

The orchestrator architecture will have its foundations on the OSGI framework already developed for the pilot project, since it has shown robustness, modularity and versatility. This will be the reference architecture for the entire SHIELD framework, enriched with innovative middleware services (secure discovery, trusted composition, monitoring and filtering, etc.).

This architecture is similar to a traditional middleware for service provisioning, with the main different that it includes a repository for the semantic models developed in Task 5.1. As already outlined in pSHIELD, in fact, the traditional service registry database will be coupled with a semantic repository in which measurements, policies and other SPD relevant information are inserted by the monitoring engine to feed the orchestration engine and the security agents and to allow them to take “intelligent” decisions.

This will constitute an important improvement in middleware for SPD applications.

4.2.3.2 Choreography: a more intelligent manner to compose services

The Choreography Working Group at W3C is developing the Choreography Description Language (CDL). CDL is designed to complement BPEL and other extended Web services technologies by defining the executable processes needed to implement a piece of a business choreography or operation process for one particular system operating multiple devices.

While BPEL’s (Orchestration based) abstract processes are intended for process interactions, CDL provides additional capabilities beyond BPEL, including how different process engines might talk to each other, such as systems of systems scenario involving a BPEL engine on one side and a choreographic engine[14] on another.

Metadata such as the element name (embedded system id), security information to be shared, policy information on non-repudiation, acknowledgment policy, human approval policies, and so on cannot be defined with BPEL alone when BPEL is used in conjunction with other technologies in a wide-process collaboration across multiple systems, such as in an extended systems of systems scenario where multiple devices are collaborating and unpredictable processes might happen.

CDL is intended for use in scenarios in which it isn’t possible to define a single controlling entity for the entire interaction, and it may be necessary to define the rule for sharing control of the overall flow. CDL defines how and when to pass control from one element/agent to another.

[14] RosettaNet Implementation Framework (RNIF) and ebXML Message Service Specification (MSS).

"CDL provides a "global" definition of the common ordering conditions and constraints under which messages are exchanged that can be agreed upon by all the Web services participants involved in the interaction. Each participant can then use the global definition to build and test solutions that conform to it. Systems may not want to cede control to an external component process engine and may want to use CDL for negotiating a smooth handoff of control. CDL is not an executable language but rather a declarative language for defining interaction patterns to which multiple parties can agree."

W3.org

The syntax of the package construct is as follows:

```
<package name="SupplyChain"  author="Ericn" version="1.1"
targetNamespace=www.iona.com/artix/examples  xmlns="http://www.w3.org/2004/04/ws-chor/cdl">
          importDefinitions*
          informationType*
          token*
          tokenLocator*
          role*
          relationship*
          participant*
          channelType*
          Choreography-Notation*
</package>
```

The package model defines the participants in collaboration and their respective roles and relationships. Once agreed upon, packages are exchanged a cross trust boundaries among the collaborating companies to share explicit definitions.

The package construct allows aggregating a set of choreography definitions, where the elements informationType, token, tokenLocator, role, relationship, participant, and channelType are shared by all the choreographies defined within this package.

4.2.3.3 Orchestration vs choreography

In this section, we'll compare and contrast the different approaches to using WS-BPEL to define the flow. In general, there are two approaches to implementing the same business process:

- Orchestration-centric.
- Choreography-centric.

These approaches are described in further detail in the following subsections.

Orchestration-Centric Approach

The following example shows the WS-BPEL pseudo-code for the OpenAccount process (this pseudo-code shows the essential logic of the Web services orchestration using WS-BPEL keywords but without the XML syntax):

```
receive 'OpenAccountRequest'
invoke CollectAccountInfo
invoke ValidateAccountInfo
assign AccountInfoInvalid = ValidateAccountInfoResponse
          while AccountInfoInvalid = true
                    invoke RepairAccountInfo
                    pick onRepairAccountInfoCB
                              invoke ValidateAccountInfo
                              assign AccountInfoInvalid =
                              ValidateAccountInfoResponse
                    otherwise // timeout - assume AccountInfo can't be
                    repaired
                              invoke DeclineAccountApplication
                              terminate
                    end pick
          end while
invoke OpenAccount
invoke SendConfirmation
```

In this approach, the OpenAccount process is implemented as a WS-BPEL orchestration that has the following characteristics:

- The orchestration is initiated when an OpenAccountRequest is received.
- The orchestration invokes Web services (such as OpenAccount and CollectAccountInformation) to fulfill the steps in the orchestration.
- The orchestration defines the business logic for the OpenAccount process, including control logic (e.g., control of the while loop that handles invalid account information) and data flow.

[…] • context awareness features to refine and extend the existing middleware orchestration functionalities in order to improve their performance (strict liaisons with Task 5.4); […]

Choreography-Centric Approach

In this approach there is no a single engine that takes the control of the entry point. So that interactions are peer-to-peer. Control is not established by neither of the entry nor sequential points that is being processed. This enables the capacity to dynamically reorganise and be tolerant to unexpected happenings.

Comparing Orchestration-Centric and Choreography-Centric Approaches

In general, the choreography-centric approach is more loosely coupled. One key difference is that the orchestration approach assumes a single, central point of control over the entire scope of the process execution, while the choreography approach assumes that execution control is shared, potentially across multiple systems process engines or various other technologies.

In a heterogeneous and semantically differentiated environment like nSHIELD, choreography mechanism is a valuable attribute that should face the unexpected threats. Therefore, Choreography should increase fault tolerance and thus system' resiliency.

4.2.3.4 SPD core State chart (SCXML)

Some SPD functionalities might require a formal structure of sequencing. This formalisation is brought by SCXML standard with formally expresses a state machine through xml language. This is a nice solution for regular and deterministic use cases that might derive from nSHIELD project.

An example of xml notation for SCXML:

```
<?xml version="1.0" encoding="UTF-8"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0" initial="ready">
    <state id="ready">
        <transition event="watch.start" target="running"/>
    </state>
    <state id="running">
        <transition event="watch.split" target="paused"/>
        <transition event="watch.stop" target="stopped"/>
    </state>
    <state id="paused">
        <transition event="watch.unsplit" target="running"/>
        <transition event="watch.stop" target="stopped"/>
    </state>
    <state id="stopped">
        <transition event="watch.reset" target="ready"/>
    </state>
</scxml>
```

4.2.4 SHIELD data and metadata management

[…] A complete architecture will be defined for developing the SHIELD core SPD services, which will be based on the exchange of semantic metadata, used to describe, for each service, its model, the relevant security requirements and the needed SPD components. […]

[…] The designed semantic language may be used also to represent profiles and policies according to interoperable and self-describing formats. The exploitation of semantic technologies will allow meaningfully representing and reasoning about context and policy information. […]

[...] The benefits brought by semantic technologies developed in Task 5.1 will be also adopted to exploit the security agent capability and adapt security needs and associated policies to possible unforeseen situations.

The main outcome of this task will be the development of a software prototype (on M18 and M30) ready to be

[…] The security agent will be designed to interpret and elaborate the SPD information generated by the SHIELD multi-layer framework. So the Security Agent produces high-level SPD information which are aggregated and eventually shared and distributed with other Security Agents acting with different scopes to the SHIELD systems. The high-level SPD information will be assessed with the metrics defined in task 2.2, in order to assess the SPD level of the single layer as well as of the overall system. […]

As anticipated in the previous chapter, the SHIELD middleware will be enriched with new semantic models to capture all the SPD relevant information. For that reason the (semantic) knowledge repository in the middleware layer will contains, together with the OWL-S entries (whose adoption is confirmed in the nSHIELD project), also an heterogeneous set of data and metadata, including:

- *Policies* to drive the composability
- *Context information* to model the system and optimize the control
- *SPD attributes* for metric evaluation

This information will be used by the security agent to take proper decisions on composability, as will be described in the overlay section (see Figure 4.16).

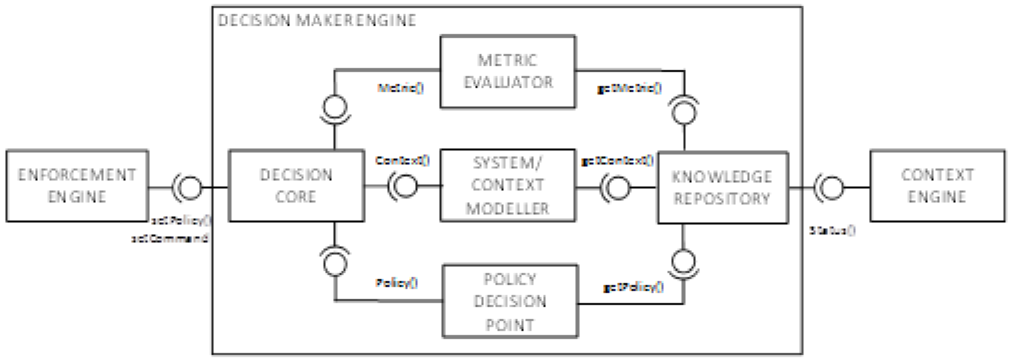


Figure 4.16: Knowledge base for the SHIELD middleware and overlay

So, the architectural choice in data and metadata management for the nSHIELD project has been to <u>move the interpretation complexity</u> of knowledge base management from middleware core services to overlay context engine.

In particular the process has been decoupled:

- the middleware services will collect the information and populate the data bases, while
- the overlay will retrieve from these data bases only information relevant to the subsystem under its control.

This is necessary since the middleware has significant capabilities and hosts services and adapters to interface the external world (and to exchange information), while the overlay has the "intelligence" to use these data, but not the computational capabilities of managing them.

In the following section a new middleware core service, for data collection and data analysis for intrusion detection purposes, will be introduced.

4.2.5 SHIELD monitoring, filtering and intrusion detection service for interface protection

[…] The interaction between SPD-middleware and ES nodes will be bidirectional. The SPD-middleware will use data received by ES nodes and will provide information to the upper layers of the system. In both cases information passing through the middleware (e.g. information, configurations and data) should be represented in a proper way (e.g. /by using semantics) in order to enable features for providing advanced service discovery, composition and orchestration functionalities to the applications. […]

A monitoring and filtering service will be designed to allow data collection. Middleware is indeed the collector of the information generated by node and network devices, since it hosts the information repositories used by the overlay to perform dynamic composability.

As information collector, middleware interacts with the external world and is subject to several threats that must be faced by a new core service: the Intrusion detection service (that performs also the above mentioned monitoring tasks). An overview of the potential threats is provided in the following.

4.2.5.1 Threats related to SPD services

4.2.5.1.1 Network/Transport level related threats: Overload and DOS/DDOS attacks

In this chapter we give an overview over the basic expressions and definitions related to overloads in mass servicing systems.

In the context of nSHIELD middleware, all service interfaces available for outside components may suffer from the problem situations described below, and considerations listed should also be analysed and protection measures may possibly be implemented. All services publicly available on open networks, like discovery and possibly composition are concerned. A proposed solution for protection is described in chapter 4.2.5.2

4.2.5.1.1.1 Nature of Overload

Overloading of information systems or servers may result in denial of service or in unacceptable response times. Overload can occur basically because of the following reasons:

- *Natural overload* during the normal operation of the system, caused by peak hours (e.g. restart/installation of a sub-system with many devices), spam, congestion in networks etc. In this case, overload can be considered as a result of improper design of the system from performance point of view (insufficient resources), or the lack of a proper rejection/fallback strategy.
- *Result of a malicious attack* (referred to as denial-of-service or DoS attack). Many past cases of synchronized DoS attacks against servers of leading information technology companies show that DoS attacks can easily paralyze Internet services for several hours. These types of attacks are especially dangerous since they require minimal preparedness from the attacker and the consequences are very damaging. Protection against DoS attacks is very hard if an attacker starts the action by penetrating into many weakly protected computers all around the world, use them as zombie systems and execute a synchronized attack against the target (as happened in many cases [64], [65], [66]). Similar attacks can be relevant to all interfaces of nSHIELD, which are accessible via internet or via public networks routed to internet without proper firewall protection.

As it turns out, protection against overload is one of the most crucial questions in the design of dependable servers. While in case of natural overload, the question is how to reject requests to allow others to be served; in DoS protection we should assume that a malicious attacker is intelligent and knows the applied protection method and will try to cheat it. Malicious attacks can produce extreme, improbable situations, so a protection method should work not just in natural-like overload cases but in any extreme situations as well.

Another serious threat is the blocking type attacks, which can be handled only by controlling the request admission function and detecting blocking. As long as this threat is not avoided, other protection methods would be useless. Therefore, the DoS protection should face each of the following overload and denial-of-service type threats:

- **Huge request**: one or a few requests, which occupy too many resources, can cause denial-of-service of the server.
- **Blocking**: a malicious attacker can very easily block an operation by exploiting the fixed limits (e.g. number of threads) or bugs in the server. Even though this case can be explained as an overload of resources, but detection of this type of events is significantly different from other overload situations.
- **Flood**: an attack aims to overload the server by generating an unbearable number of requests. In this case the overload can be much bigger than a natural overload can cause.

Flood can be handled at different levels based on its intensity:

- **Overload**: the server gets more requests than it can handle, but it has enough capacity to select and reject unwanted load and serve the remaining ones. (Optionally rejected requests may get a polite rejection message.)
- **Moderate flood**: Buffers get filled, packets get lost, but the server remains operational. Those requests will be served, which can get through, but the server does not have the capacity to select unwanted requests. (Preferably polite rejection mechanism should be disabled in this case.)
- **Massive attack**: Incoming messages overload the operating system under the server and totally block its operation (no served requests). These attacks usually use distributed zombie systems to flood their targets (DDoS).

4.2.5.1.1.2 Performance Considerations

It is very important to clearly understand what can be the goal of an overload protection subsystem. From the client's point of view two major factors are important: *maximum response time* and *rejected request ratio*. One may think that the average response time can be also a considerable factor; however we think it has a significantly less importance, as it will be explained later on.

From the server's point of view the aim is to *stay alive* in any cases (avoid final denial-of-service), *utilize system resources most effectively* to maintain high performance (especially CPU utilization should be 100% in case of overload, while blocking should be avoided).

To control the above performance factors we have basically two performance controlling values: the *length of the queue* and the *number of server threads*. However we will see that performance considerations in some sense contradict security considerations.

The above considered performance factors will be analysed during the nSHIELD research project from many points of view:

- **Measurements**: operation of the proposed solution will be measured in realistic environment. nSHIELD SPD metrics relevant to this case will be developed.
- **Queuing Theory**: theoretical modelling and analysis.
- **Simulation**: if the mathematical analysis of our proposed solution is not available, we will use simulation technique to determine more precise results.

4.2.5.1.1.2.1 Response Time, Rejected Request Ratio

From the client's point of view the response time and the rejected request ratio are the most important factors. If an adequate overload protection is not applied, response times may increase above a limit where users (in this case, nSHIELD-enabled devices) could time out waiting for the response and therefore this request can be considered lost and, on the other hand, server resources were consumed uselessly. This negative feedback can drastically degrade the performance of the server: response times continue to increase and the ratio of lost requests will increase as well, finally resulting in a virtual denial-of-service.

The response time and rejected request ratio can be handled by buffering or queuing. In case of burst in the load traffic the requests can be served only up to a certain limit. A server has a finite service capacity and if it gets higher load than this limit, at least the extra part of the incoming traffic should be rejected to limit the response times. So if a server gets a 110% load for a long time, at least 10% of the requests should be rejected. Unfortunately, because real traffic has a remarkably big variance (bursts), the server will drop requests even under the theoretically handleable 100% load. The ratio of dropped requests in this case depends on the time while requests wait in the queue. The longer they wait, the longer bursts can be handled without unnecessary rejection. *Thus the rejected request ratio can be considered as a function of response time*. The traditional way to limit response times is to limit the number of requests in the system (limit the length of the queue). We will see that security will require a more adequate mechanism to control response times.

Based on the theoretical modelling we can describe the operation of a server based on only some parameters. This description is almost independent from the used overload protection method.

- **Below 90% load**: queue is almost empty. Rejected request ratio is very small if the queue is longer than a minimal size (10).
- **Between 90% and 110% load**: behaviour of the server in this interval greatly depends on the applied overload protection technique. Rejected request ratio is somewhere between 5% and 15%.
- **Above 110% load**: the queue is almost always full. Response times go up to their limit. Request rejection ratio will be close to the magnitude of the overload (load-100 %).

As we can see from the above, the applied technique is important only in the 90%-110% load interval. We should focus on the acceptable response times and try to avoid unnecessary rejections in this interval.

4.2.5.1.1.2.2 Performance and Utilization

Since a server is a complex system its modelling is not straightforward. Thanks to multitasking, one computer may serve more than one request in the same time. In parallel executions threads can be overlapped without remarkably affecting the execution of each other, but if too many threads are started together, the overall performance will not increase. This *thread limit* depends on very many factors (request type, hardware configuration, network bandwidth, speed of disk I/O, etc.). Although we can explain the meaning of this limit theoretically, it is very hard to determine it in practice, because its value may greatly vary during operation.

That's why we proposed another approach for controlling the performance of the server. Namely, if the utilization of the server resources is the highest possible, the performance of the server should be the highest possible as well. In other terms, *the utilization is the highest if no resources are wasted* when they could be utilized.

4.2.5.1.1.2.3 Arrival Rate

Investigation of mass-servicing systems shows that if the number of users is high, requests arrive according to a memoryless process, named *Poisson-process* in almost every case. In this case the time intervals between incoming requests are exponentially distributed. The parameter of this exponential distribution is referred to as *arrival rate* and usually denoted by λ. Since the mean value of an exponential distribution is $1/\lambda$, the arrival rate can be interpreted as the average number of requests within a time unit or $1/\lambda$ can be considered as the average elapsed time between two request arrivals.

The Poisson process models the reality very well, so we used this load characteristic in each case.

4.2.5.1.1.2.4 Service Rate

To enable the simple modelling of server systems it is very important to hide the internal structure of a server, it can be considered as a black box and we need only one performance-measuring factor to describe this system, which is the service rate. This service rate gives how many requests leave the system in one time unit in average. Of course if we want to be more precise, not just the rate, but the distribution of the leaving requests should be considered. Further exponential, deterministic and general distributions will also be analysed.

One should also understand that the *service rate is independent from the response time and the servicing time*. Imagine the case of a pipeline system in a car factory. In each hour a car leaves the factory, but it takes four days for a car to be prepared. In this example the service rate is 1/hour, the service time is 4 days (96 hours), while the response time (together with the waiting in the queue) can be as high as several weeks.

For a system with limited capabilities this service rate has an upper limit, which is denoted by μ. This μ reflects the performance capacity of a server. In case of computer systems the maximum service rate is approximately equal to the reciprocal of the average CPU usage time of a request.

4.2.5.1.1.3 Security Considerations

After the usual performance measures of queuing systems we overview the security considerations as well, where we also have to assume malicious intent to cheat the protective mechanisms.

4.2.5.1.1.3.1 Blocking Avoidance

The *limitation of server threads* according to a fixed value can lead to the possibility of *blocking attacks*. Note that if this thread limit is increased to any fixed value an attacker always can use this technique to block the actual server.

To avoid blocking, *the server should not have (small) fixed limitations on any resources*, otherwise a malicious attacker may issue some requests which consume these small amount of resources, hold them allocated and this way block the service of other requests.

4.2.5.1.1.3.2 Recognizing Malicious Flood

The *fixed limitation of the length of the queue* can prevent the DoS protection subsystem to reject malicious requests (black list) effectively and admit correct requests (white list), since if one determines the maximal length of the queue based on performance considerations then, even if malicious requests can be spotted *too many correct requests will be dropped at the end of the queue*, so an attacker can reach his goal to deny many good requests and thus virtually block the server.

On the other hand *if the allocation of a resource is not limited* (either the length of queue, or the number of threads) a malicious flood or even a natural overload can lead to *unlimitedly growing response times.* This is unacceptable and practically means a denial-of-service from the clients' point of view.

At first sight there is a contradiction here between performance and security considerations. The DoS protection subsystem needs a long run from a flood to recognize the malicious requests more precisely and preferably no requests should be dropped during it, while performance considerations call for a high request rejection rate to maintain response times. Our proposal solves this problem by letting the queue grow and rejecting requests at the beginning of the queue.

4.2.5.1.2 Presentation/Application level related threats

4.2.5.1.2.1 Programming bugs

Common vulnerabilities caused by typical security-relevant programming bugs in the implementation make up a sizeable part of software vulnerabilities, which are worthwhile to target when using automated testing and security analysis. In the current case, the implementation of middleware core SPD services are proposed to be tested against a list of bugs, such as:

- Buffer overflow errors
- Integer overflow errors
- Format string bug
- SQL injection (injection errors related to database access)

4.2.5.1.2.2 Cross-site scripting vulnerabilities

Cross-site scripting (XSS) is vulnerability where malicious scripts are injected into vulnerable Web pages, causing the user's browser to execute malicious code [67]. In a Web service context, XSS vulnerabilities are triggered indirectly – a Web service could feed malicious content to a web application, which in turn would present this content to those who used the web application.

4.2.5.1.2.3 XSLT vulnerabilities

XSLT is a content-generation language using select and merge operations. The language is Turing-complete and can use various extensions from the underlying environment. In the recent years, several vulnerabilities were found in various XSLT parsers [68] and XSLT usage scenarios. For example: if the XML parser supports XSL transformations within the <Transform> tag of an XML signature, it may contain XSL injection vulnerability [69].

4.2.5.1.2.4 DTD parser implementation vulnerabilities

DTD entities are resolved by the XML parser when doing initial parsing, and DOM parsers attempt to store them in memory. If the attacker uses a sufficiently deeply nested DTD entity chain, it can cause the parser to crash or consume all of a system’s available memory [70].

4.2.5.1.2.5 Coercive parsing vulnerabilities

In order to exhaust the system resources of the web service, the attacker can send an XML message containing a large number of – or very deeply nested – entries [71]. Depending on the web service's XML parser implementation, the parsing of such an XML message may cause abnormally high CPU load and memory consumption or even crash the web service.

4.2.5.1.2.6 SOAP specific vulnerabilities

Some special SOAP features and functionality can be implemented in a vulnerable way. The attacker may create malformed SOAP messages in order to cause denial-of-service against the local or remote site, for example by creating large SOAP arrays [72] or modifying the <ReplyTo> element in the header [73].

4.2.5.1.2.7 External reference attacks

If a reference is identified as an URI, such as the <RetrievalMethod> in XML signature, the specified URI may refer to remote content and the attacker can cause a denial-of-service attack against the remote site indirectly, or can force outbound network connectivity by using URL schemes (e.g. ldap://, file://, ftp://) [74].

4.2.5.2 Protection against threats: intrusion detection service (DoS / DDoS protection)

The proposed solution of protecting nSHIELD Middleware Core SPD services is to add an SPD service that is monitoring vital server resources and utilization of the Core SPD services being monitored.

The proposed solution can be seen in Figure 4.17, with the example of the Discovery Core SPD Service being protected. The DoS Protection Subsystem is described in chapter 4.2.5.2.1, while chapter 4.2.5.2.2 discusses how the nSHIELD system could provide DDoS protection on the system level.

The Intrusion Detection Bundle would thus be implemented as an input filter for other Core SPD Services with public interfaces.

> *[…] This task will also define and implement specific interfaces (based on the design results of task 2.3) for accessing middleware capabilities from outside the system. The SPD modules will be implemented as software modules which will become part of the prototypes delivered by WP5 on M18 and M30. […]*

The type of the interface supported by the Intrusion Detection Bundle would be defined on the transport protocol level (e.g., SOAP messages encapsulated in XML messages, served by HTTP server services). All other Core SPD Services would be able to dynamically set up filtering of their inputs using the input filtering services of the Intrusion Detection Bundle.

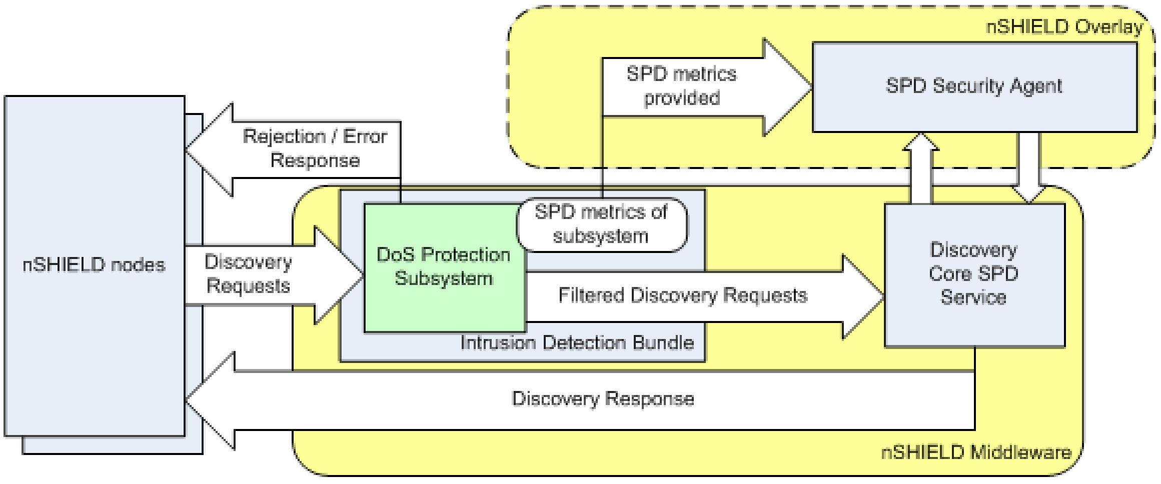


Figure 4.17: The logic block structure of the Intrusion Detection Bundle

4.2.5.2.1 DoS Protection Subsystem

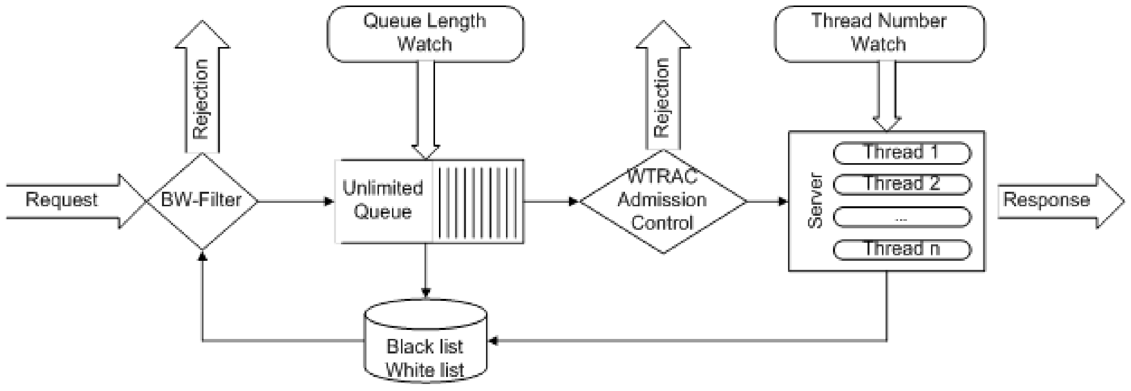


Figure 4.18: The Logic Block Structure of the DoS Protection Subsystem

After a careful evaluation of problems and possible solutions related to overload, DoS and blocking threats we designed a pretty good subsystem, which can protect against these threats with a combined mechanism. According to our initial discussions this solution seems to be optimal not just from protection, but also from performance point of view.

The heart of our protection concept is the *Waiting Time based Request Admission Control* (referred to as WTRAC), which evaluates each request and decide whether it can be admitted to the server for processing or should be delayed or should be rejected. In practice it reads one request from the *unlimited queue* and waits until the server will have enough resources to start the processing of this new request.

WTRAC solves the problem of unlimitedly growing response times, which is a consequence of the unlimited length queue (which could cause blocking) by observing the waiting times of the requests in the queue. If a request gets so old that its service cannot be expected within a given time limit (waiting time + processing time > time limit) the request is rejected.

Using this WTRAC technology we got a *self-calibrating method* to dynamically determine the *number of started threads* and to *limit the response times* without applying undesirable fixed limits within the architecture. WT*RAC provides ideal performance behaviour* in case of natural overload: *rejection ratio is the smallest possible*, response times are limited and their variance is also minimal. In case of *blocking attack* or *flood* the malicious characteristic of the load *can be very easily recognized*: the number of threads increases in case of blocking attacks or the length of the queue goes above a high limit in case of flood. Simple *monitoring mechanisms (Queue Length Watch, Thread Number Watch)* can be applied to handle these attacks as shown in the figure.

In case of natural overload the average length of the queue will be controlled by the maximal waiting time and will tend to an optimal, self-calibrating value, which means 10-100 records in practice.

In case of flood, when the server receives much more requests than it can serve (i.e. request rate is more than double of the service rate) the average length of the queue grows up, but remains limited. In such case the server can process only a part of the requests, but remains operational with 100% load.

In case of malicious denial-of-service attack, the length of the queue grows without limit, what the Queue Length Watch can detect. In this case the queue is full with malicious requests, out of which only a small number (i.e. less than 10%) belongs to normal queries, all the others come from malicious sources. An intelligent algorithm can distinguish then between normal and malicious requests and put the bad sources on a black list. If filtering based on the black list is not enough to maintain the normal operation of the server, the system can turn to white list based filtering, when only those requests will be served, which behaved normally in the past.

In case of blocking a similar effect can be observed in the number of started server threads. If their number increases unnaturally, the Thread Number Watch can notice it and put the blocking requests to the black list.

In summary, our proposed solution is a very promising protection method against both natural and malicious overloads and blocking, while on the other hand it does not even degrade the performance of the server, but provides a much better behaviour than the current solutions.

4.2.5.2.2 DDoS protection - Connection to nSHIELD Overlay

The above protection technique can provide good local response to DoS attacks. However in case of heavy distributed DoS attack we can protect only at the network level, which means the Intrusion Detection Bundle would have to alert the Overlay system of nSHIELD about the fact of the attack and back propagate the black list or in more severe cases the white list. With the use of SPD metrics provided by the DoS protection subsystem and reconfiguration capabilities implemented in the control algorithm of nSHIELD Overlay layer, it would be possible to provide an unmatched protection against these serious attack methods.

The SPD metrics that the DoS protection system provides in order to realize higher-level system protection would include the following:

- Wait and processing times for Services protected
- Queue length of Service
- Thread count of Service
- Amount of allocated resources
- Number of rejected service requests (cumulated and per interval)
- Overload detection status (load % and/or internal state of protection)
- Blacklisting / Whitelisting events of Nodes or Blacklist/Whitelist

The SPD Security Agent in nSHIELD Overlay would be able to utilize the above metrics in SPD level decisions and specifically, malicious or erroneous behaviour of nSHIELD Nodes, which is not reported or recognized using the SPD metrics reported by the nodes themselves, could be detected and taken into account for system composition.

4.2.5.2.3 Middleware core SPD services vulnerability testing and countermeasures

To make the Middleware functionalities implemented in nSHIELD more secure, security testing regarding public service interfaces is proposed. Thus, the Core SPD services providing system-wide functionality to components on other levels of the nSHIELD system, such as nodes and network components will be made more securely integrated, and the whole system less error-prone.

To detect potential vulnerabilities originating from the implementation, we propose security testing of the external interfaces of the nSHIELD middleware, supplementing the Core Services Certification (see Chapter 4.2.7).

Common security vulnerabilities may be detected in various ways, from source code analysis to black-box testing. As source code analysis is a design-time technique and we are evaluating the external interfaces of composite services, we focus on detecting vulnerabilities through the use of active testing methods – specifically fuzz testing. In this section we briefly describe the most important fuzzing-related technologies.

Most traditional security testing methodologies focus on finding evidence of known vulnerability types. Fuzz testing approaches the same problem from a different angle – instead of trying to locate specific vulnerabilities, it attempts to manipulate the input to the target composite service (the Target of Evaluation – ToE) in order to discover previously unknown typical vulnerabilities.

According to [75], *(fuzzing is) a highly automated testing technique that covers numerous boundary cases using invalid data (from files, network protocols, API calls and other targets) as application input to better ensure the absence of exploitable vulnerabilities. The name comes from modem applications' tendency to fail due to random input caused by line noise on "fuzzy" telephone lines.* Fuzzing is not purely a security testing technique – it has numerous applications in quality assurance (QA) processes as well (specifically robustness testing). Even there it focuses on identifying bugs arising from input validation issues, and does not validate the actual business logic implementation; fuzz testing cannot substitute for functional testing. Fuzz testing also tends to be binary in the sense that it only evaluates whether a certain test vector causes the ToE to crash, lock up, or act in a way that is easily identifiable as non-conformant – it typically does not attempt to identify the cause of the crash in detail. As this also means that typical fuzz testing produces no false positives, it makes fuzz testing very useful for nSHIELD; vulnerabilities reported by such a 'binary' fuzz tester are guaranteed to be valid, and will not degrade the overall trustworthiness of the nSHIELD platform.

4.2.6 Adaptation of legacy systems

[…] This task foresees also the design of highly-dependable interfaces and/or, adapters and/or enablers to make heterogeneous legacy SPD solutions (protocol, standards, mechanisms, techniques, etc.) for ES nodes, networks and middleware interwork transparently with the enhanced capabilities provided by the SHIELD approach. The main outcome of this task will be the design of adapters to make legacy devices capable to support the SHIELD SPD-functionalities, as well as the development of some prototypal software. […]

In order to allow non-native SHIELD (i.e. legacy) devices to be integrated into the SHIELD framework, it is simply necessary to enable them with the possibilities of being *discovered* and *composed*. This can be done by developing specific adapters (HW or SW) that contain the semantic information necessary for the discovery/composition procedure and that are able to communicate them. Usually it is done by means of pluggable web server (HW) or ad hoc software routines (SW).

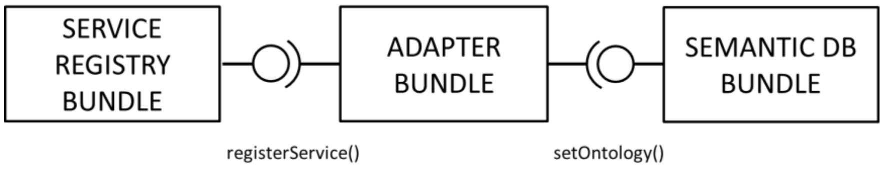


Figure 4.19: SHIELD generic adapter

In Figure 4.19 the architecture of a generic SHIELD adapter are highlighted. The main interfaces are:

- The possibility to register the provided Innovative SPD Functionality in the Service Registry;
- The possibility to publish the semantic description of the Innovative SPD Functionality in the Semantic DB;

4.2.7 SHIELD middleware protection profile certification

A Protection Profile[15] (PP) is a document used as part of the certification process according to the Common Criteria (CC). As the generic form of a Security Target (ST), it is typically created by a user or user community and provides an implementation independent specification of information assurance security requirements. A PP is a combination of threats, security objectives, assumptions, security functional requirements (SFRs), security assurance requirements (SARs) and rationales.

A PP specifies generic security evaluation criteria to substantiate vendors' claims of a given family of information system products. Among others, it typically specifies the Evaluation Assurance Level (EAL), a number 1 through 7, indicating the depth and rigor of the security evaluation, usually in the form of supporting documentation and testing, that a product meets the security requirements specified in the PP.

For the nSHIELD project, the possibility of editing a protection profile for the SHIEDL middleware will be explored, depending on the completeness of the architecture. If possible, this profile will be certified and recognized at European Level as the first certified SHIELD component. This will open the procedure for the certification of SHIELD compliant Embedded Devices.

[15] Source wikipedia and Common Criteria portal

5 SHIELD policy based management assessment

5.1 pSHIELD results and adopted technologies

5.1.1 pSHIELD policy based management architecture

In the pilot project, a typical PBM architecture has been mapped into the SHIELD general architecture. The latter includes two types of nodes at the button node layer that are categorized based on their capabilities in terms of processing power and capacity, i.e., power nodes and sensor nodes. Power nodes are described to be more resourceful while sensor nodes are typically seen as resource constrained devices. Upper supporting layers constitute network, middleware and application layers while agents in a vertical overlay monitor/tune those layers. Given the aforementioned architecture, a PDPs and PEPs from a typical PBM architecture can be mapped naturally to power and sensor nodes respectively. Figure 5.1 presents the proposed PBM mapping.

On the lower layer, sensor nodes being the managed resources are considered as policy enforcers, i.e., PEPs. The latter, based on the XACML model, should enforce authorisation decisions and handle affiliated obligations specified by applicable rules. PEPs can support local policy storage in order to comply with COPS-PR mode of operation hence the provision of a local PIP although not compulsory. However, this depends on the capabilities of deployed sensor nodes whether they can afford a form of local policy storage and decision making. Moreover, power nodes are those nodes that are more resourceful than the sensor nodes which make them natural decision making points able to process/translate policies and deduce rules to be enforced by affiliated PEPs. The COPS protocol can govern the communication between PDPs and affiliated PEPs but not exclusively as SNMP is an option as well (where an LPIP is no more required).

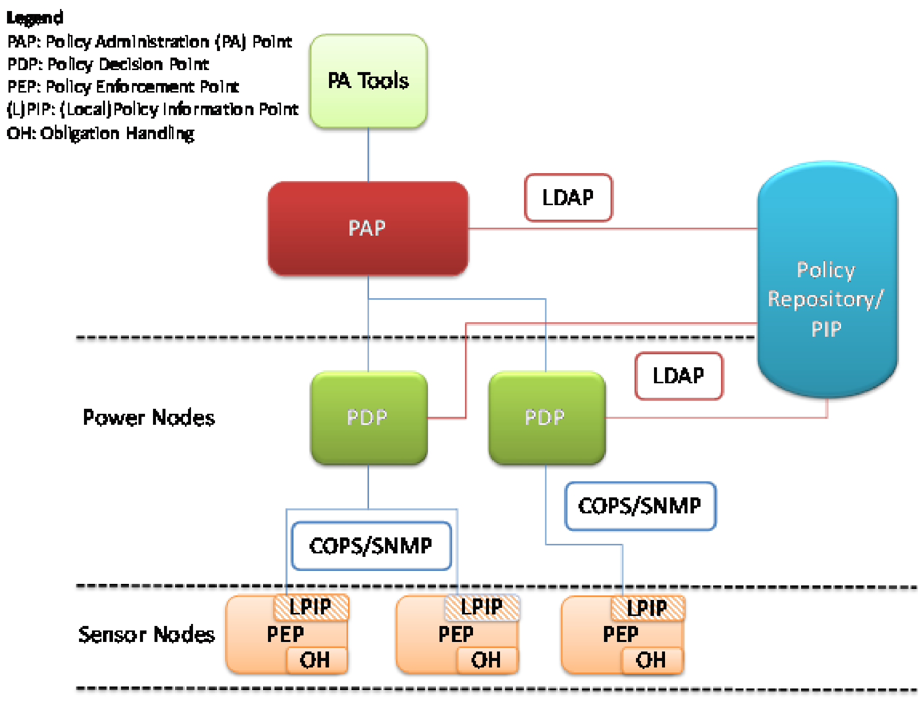


Figure 5.1: PBM Mapping

A group of PDPs can access the repository of policies, (i.e., PIP) in order to retrieve needed polices for evaluation. This is done through LDAP that is a protocol suited for lightweight read-intensive operations allowing for directory access from different platforms and locations. The policy repository is managed solely by the policy administrator point (PAP). Also, PAP is responsible for providing policy authoring tools besides management and control capabilities. These could include creation, termination, activation, listing, amending and synchronizing policies. Commercially for instance, Cisco's PAP (Cisco) manages, administer and monitor policies in a central manner compatible with LDAP. It also provides several features such as web-based editing and composing of policies in a drag and drop manner, policy scope definition, delegation management and aiding functionalities for understanding the implications of policy changes. Concerning, LDAP implementations, an open source version is available as mentioned earlier at (OpenLDAP, 2011). However, COPS does not seem to have an open source implementation where some commercial ones could be available. If SNMP is considered as an alternative for COPS, some open source and free implementations are available such as in (NET11). XACML is indeed the policy specification language to be used as argued in earlier discussion.

Concerning pSHIELD main scenario where a monitoring and access control system is put in place to oversee rail-transported hazardous materials, the above PBM is considered suitable. Locking and access control mechanism in addition to installed sensors can be seen as PEPs where the central control unit in the train carriage can be seen as a PDP with local access to PIP. Moreover, the central command centre overseeing the operation of the monitoring system is seen as a PAP with policy administration tools and repository support. The PIP is expected to be distributed which allows a given PDP to access it locally where a PAP can manage such a distributed PIP through LDAP.

As a final step, a performance evaluation of the policy execution (as PDP) is reported, with specific focus on the computation time during policy execution.

The processing time depends on:

- Policy specification language (high level or low level)
- Type of policy execution engine
- Underlying formalisms used to describe attributes (e.g. Knowledgebase in pSHIELD)
- No. of simulteneous execution

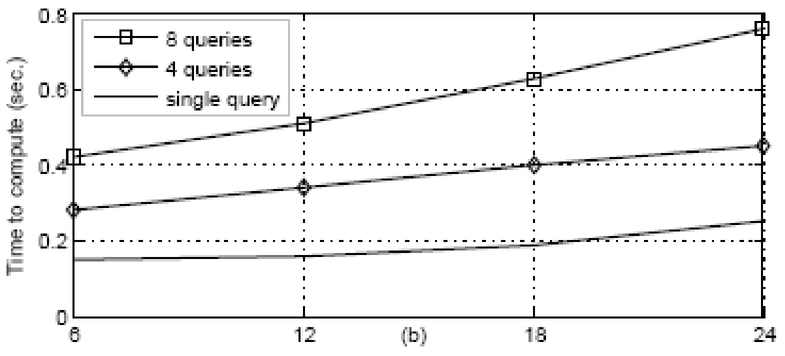


Figure 5.2: N° of instances/class in Knowledge Base

Figure 5.2 shows the computation time for simultaneous policy execution with the following simulation parameters:

- Rule-based semantic policies (SWRL-SQWRL)
  - *A high level language*
- Underlying formalism: OWL
  - *High level compared to simple XML*
- Simultaneous queries from Application requires execution of simultaneous policies
- Performance measure in P4 2.0 GHz,1GB RAM windows machine

Implications for pSHIELD: Latency is one of the QoS requirements for Web services at Middleware level; latency includes request processing time. Figure shows that even with small no. of instances simultaneous policy execution takes increasing amount of time.

For that reason run-time decision support with simultaneous query processing may not be possible with such settings.

The assessment of this technology is reported below:

Table 5.1: pSHIELD policy based management – ASSESSMENT

pSHIELD policy based management - ASSESSMENT	
Limits of this approach to be overcome	The policy based management architecture developed for the pilot project purposes has shown to be suitable for a deployment in a real application scenario. However, after the deployment, at runtime, the biggest problem is in the computational effort needed to perform the evaluation of the policies. This represents a problem in an SPD environment, especially when realtime (or near realtime) decisions must be taken. Moreover very poor policies have been used in the testing campaign, so there is still the lack of a reference SHIELD policy, based maybe on a consolidated standard or at least on a consolidated language.
Positive aspects to be preserved	The PBM architecture has been selected after a careful evaluation of the state of the art. This background analysis must be leveraged also in the nSHIELD project.

5.2 nSHIELD potential investigations

5.2.1 Proposed SHIELD policy based management architecture

This task aims at designing and developing a SPD-middleware policy-based management for ensuring a high level of security, privacy and dependability in systems composed by Intelligent ES Nodes (developed in WP3) and based on Smart Transmissions (developed in WP4) on the base of the metrics identified in task 2.2. In order to build specific management functionalities and procedures for accomplishing these objectives, several aspects will be investigated and analysed. The main ones are:

• Use of policies. Policies permit the declarative specification of security strategies separately from the implementation code of ES nodes. The use of interpreted policies allows to change the security behaviour of a node without recoding or shutting down the node;

• Design and development of algorithms and tools to enrich the smart capabilities of the middleware and increase

5.2.1.1 Use of XML-Based Technologies and Protection of the Communicated Messages

Adoption of XACML seems to be the most appropriate solution given its benefits, the standardised nature and support it enjoys as demonstrated by the analysis made on policy languages in pSHIELD.

However, in contrast to the pSHIELD architecture the following modification, also depicted in Figure 5.3, is proposed to cover different types of nodes (especially power nodes that have the capability to support PDP functionality.

1. Power Node that incorporates both PEP (typically on a node component) and PDP functionality serving requests originating from other nodes or for its own needs.
2. Power Node that is only acting as PEP, while the PDP functionality is provided by another system such as a base station.
3. Power Node that only acts as PDP serving requests originating from other nodes.

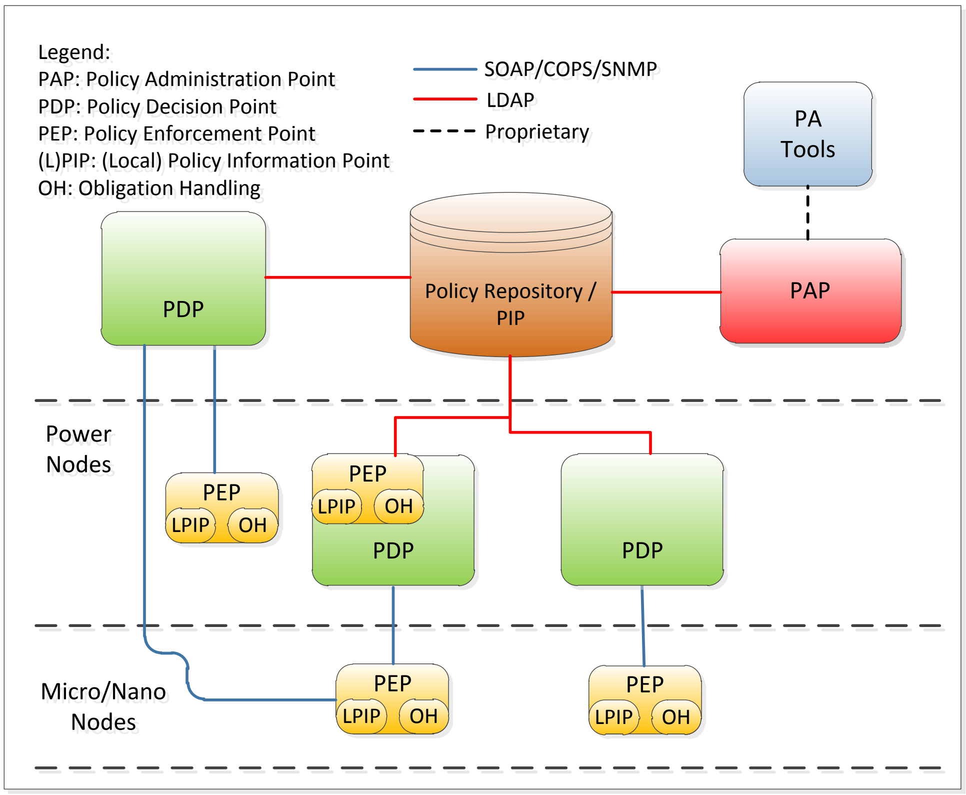


Figure 5.3: nSHIELD Policy-Based Management (PBM)

Given the dynamic nature and the desired property of self-configurability, there will be situations where there is no coordination or central control over the network of nodes. Therefore nodes that just joined the network have to discover which entity is responsible for making access decisions. In this case the corresponding systems, such as power nodes or base stations, have to advertise their capabilities regarding PDP functionality while PEPs have to be able to discover PDPs and their corresponding provided services. It is worth noting that by the term “power node” we imply a node possessing the same functionality as a micro node, but that does not have any resource constraints.

While XACML defines the structure and content of access requests and responses exchanged among PEPs and PDPs, it does not provide any details regarding mechanism(s) used to transfer these messages, thus providing the necessary flexibility to adapt to diversified environments. Protocols that have been proposed for the communications among PEPs and PDPs are COPS, SNMP while LDAP matches the requirements for accessing the policy repository and PIP. One of the issues that need considering is the protection of policy messages exchanged among PAP, PEP, PDP, and repository.

Communications between a PDP, running on a power node, and the repository can be protected using the TLS and the widely used StartTLS extension, as the power node can support the heavy computations required by TLS. However, communications among the less powerful micro/nano nodes (PEPs) and power nodes (PDP) using COPS or SNMP protocol cannot rely on the security provided by TLS or IPSec, due to their expensive computations that do not match the requirements of the constrained environment with the very limited resources [91]. Therefore, alternatives have to be adopted for this purpose.

Several lightweight alternatives have been proposed that are based on TLS/SSL, thus making them suitable for resource-constrained environments. One such approach was proposed in [92], which used ECC for key exchange and authentication, RC4 for encryption and MD5 for integrity check. According to the presented experimental results, it was able to complete a full SSL handshake within 2 seconds. Tiny 3-TLS proposed in [93] is an extension and adaptation of the TLS handshake sub-protocol, tailored for securing communications between sensing nodes and remote monitoring terminals. This protocol relies on the existence of an intermediate node, the sink node. In the nSHIELD context, the role of the intermediate node can be played by the power node.

Regarding less powerful nodes, protection mechanisms can be deployed on multiple layers including the application layer where, for instance, a proprietary lightweight cryptographic protocol that satisfies the needs and matches the capabilities of constrained environments can be used. A solution that can be adopted for this purpose is WS-Security and the corresponding mechanisms, i.e. XML Encryption and XML Signatures (enveloped, enveloping or detached) or a combination of those depending on the particular requirements to secure confidentiality and integrity of the exchanged messages. This approach is also recommended by the XACML standard.

A structured exchange of secure messages, using XML encryption and signature, for XACML messages is provided by SAML specifications (Security Assertion Markup Language) [97]. SAML is an independent platform XML standard for exchanging authentication and authorization information. SAML assertions are typically transferred embedded using HTTP or XML-encoded SOAP (Simple Object Access Protocol) messages that are transferred over HTTP or UDP [98]. OASIS has defined a profile in [95] for the integration of SAML with XACML and, among the others, the use of SAML for the secure transmission of XACML requests and responses.

If SNMP is the chosen application layer protocol for transferring XACML data, SNMPv3 comprises a set of security capabilities for network security and access control. It is worth emphasising that SNMPv3 is not a stand-alone protocol and that it should be used in conjunction with SNMPv2 (preferably). RFC 2574 defines the User-based Security Model (USM) for SNMPv3 that is designed to be secure against modification of information, masquerading, message stream modification (since SNMP has been designed to operate over a connection-less protocol) and disclosure. However, it is not able to shield against Denial of Service (DoS) attacks and traffic analysis (an eavesdropper is able to observe the traffic patterns between managers and agents).

Message confidentiality and integrity can also be provided by corresponding security mechanisms deployed at the network layer. These services will be made available to upper layers depending on the application's and other protocols (e.g. COPS) specific security requirements. In this case, the required protection for the exchanged messages shall be revisited and aligned with the mechanisms proposed and adopted at the node's network layer, based on the work carried out on WP4. If message protection at the network layer is compulsory there is no need to deploy additional heavy computations to the application layer to add one more, possibly redundant, protection layer. Nevertheless, should a web-services-oriented approach be followed, the Devices Profile for Web Services (DPWS) security framework could be of great value, as it defines a recommendation baseline for interoperable security between devices. DPWS describes the desired behaviour of clients, in terms of confidentiality, integrity and authentication is described. What is more, a web-service-based platform for WSN management was recently proposed in [94] that also featured a novel service-oriented routing protocol.

In either of the aforementioned approaches, one of the main problems related to the exchanged messages' protection is key management, especially when considering the following:

1. Communications might take place ad-hoc between nodes that do not have an established trust relationship, hence they do not (pre-) share any secrets. Dynamic structures and self-configuration capabilities demand for more flexible mechanisms.
2. Some nodes might not support public-key technologies, which further complicate the processes of establishing trust relationships and keys.

The inherent key management problems, especially in resource constraint environments have attracted a lot of attention in the research community and many schemes have been proposed within this context. A survey and taxonomy on proposed wireless sensor networks key management schemes is provided in [89].

6 SHIELD Overlay assessment

6.1 pSHIELD results and adopted technologies

6.1.1 pSHIELD Security Agent architecture

The SPD Security Agent preliminarily designed in the pilot phase is depicted in the following Figure 6.1.

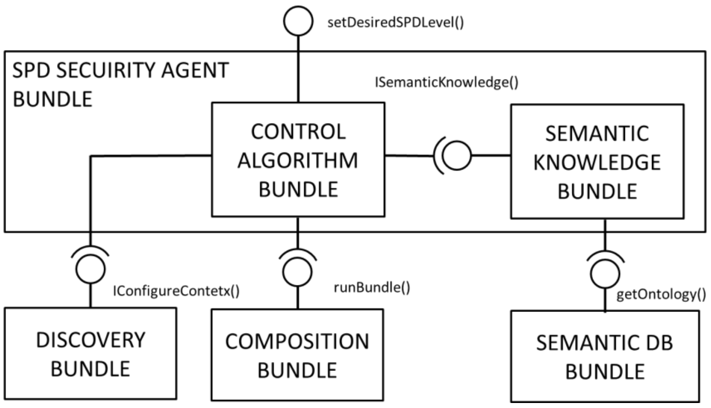


Figure 6.1: SPD Security Agent Bundle

It is composed by two elements:

- *The Semantic Knowledge Bundle:* it is in charge to get the semantic description of the available services and to make inference on their semantic model to extract the SPD level of their composition;
- *The Control Algorithm Bundle*: it is in charge to evaluate the best control strategy for the whole system in terms of proper configuration rules both for the Discovery and the Composition Bundle. The Control Algorithm can influence which services can be discovered configuring the query pre-processor and can influence the composition process limiting the composition only to the best SPD functionalities that can assure the desired SPD level.

The assessment of this technology is reported below:

Table 6.1: pSHIELD Security Agent architecture – ASSESSMENT

pSHIELD Security Agent architecture - ASSESSMENT	
Limits of this approach to be overcome	The security agent derived for the pilot purposes worked adequately in the demonstration scenario, but is far away from being a consolidated entity to be applied in all the potential domains. In particular the control algorithm module is too weak to manage the ambitious composability expected by the SHIELD framework, so a more complex (and articulated) architecture should be derived. Moreover in this architecture the security agent performs only a static composability, since it has no possibility of retrieving real time information on the underlying system. A monitoring module should be added as well, to enrich its capabilities.
Positive aspects to be preserved	The decoupling between control and knowledge management must be preserved, since the development of control algorithm and the definition of semantic models may be asynchronous, continuously evolving activities. This will increase the lifetime of the proposed solution.

6.1.2 pSHIELD Composition algorithms: Hybrid Automata approach

One of the main theoretical results of the pSHIELD project was the formalization, by means of Hybrid Automata Theory, of some control laws that are supposed to drive the SPD composition in a s way.

This concept is simple but effective: the Common Criteria approach defines a standard methodology to compose elements with precise quantification of their SPD level. Since the solution of the composition problem is not always unique, we can enrich this composition by setting further rules that allows to discriminate from one configuration to the other. This can be done by creating a dynamic model of the system and verifying, with respect to pre-defined objective functions, the most convenient configuration.

Two different approaches have been demonstrated to validate this theory.

6.1.2.1 Static Approach with Simple Optimization

The first, simple approach is based on the following steps.

At first the system "state" is identified, i.e. the set of active components (node, protocols or applications). A state is a screenshot of the system in a specific condition (for example with the node E switched on) and with the dynamics associated to this condition (for example the evolution of the node's power consumption).

The selected dynamics considered for this model constitutes the so-called context information: since the SPD is controlled via the common criteria approach, we need to insert into the model variables that could be significant to control (optimize) the evolution of the system. They could be, for example, the power consumption, the computational resources utilization, the bandwidth utilization, and so on.

The state identified in this step is depicted in Figure 6.2.

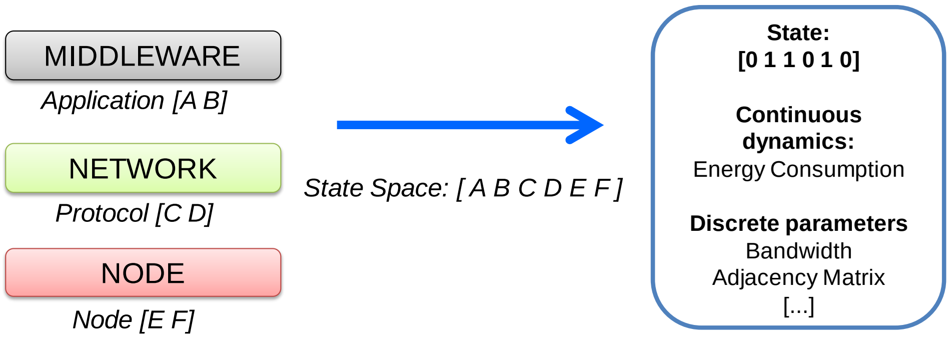


Figure 6.2: Single State representation

Secondly, different states are concatenated to obtain the universe of all the possible condition of the system: this is an enumeration of configurations. For example in a system with two nodes, two network protocols and two middleware services we 8 states (at least one component must be active).

Q = {[101010], [101001], [100110], [100101], [011010], [011001],[010110], [010101]}.

The result is depicted in Figure 6.3.

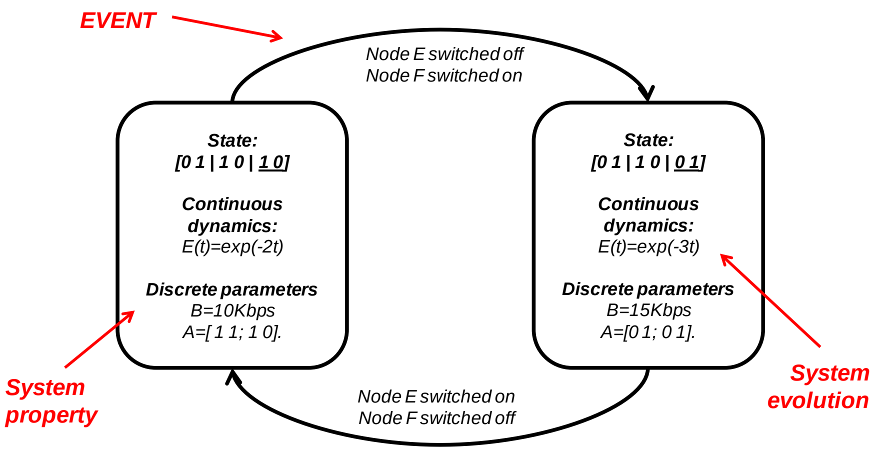


Figure 6.3: Hybrid Automata to describe all the possible configurations

The transition can be voluntary and expected (control action) or not (due to fault) but in any case each event is captured and in every moment it is possible to check the status (and evolution) of the system:

$$D = \{switch\ configuration_1,\ fault_1,\ \dots,\ switch\ configuration_n,\ fault_n\}.$$

The third step is the identification of the internal variables (and dynamics) to control. For the pilot project a simple case is considered where:

- the relevant dynamic is the power consumption of the system in a specific configuration and
- the amount of bandwidth provided by the network layer.

These variables have opposite behaviours (higher bandwidth, higher power consumption) so the purpose of the control algorithm is to choose the configuration that optimizes one of them.

This scenario has been implemented in Matlab-Simulink (see Figure 6.4) and is composed by two nodes with two different dynamics for the power consumption and for bandwidth utilization. It is important to notice that both these configurations should be valid SPD configurations (see CC approach).

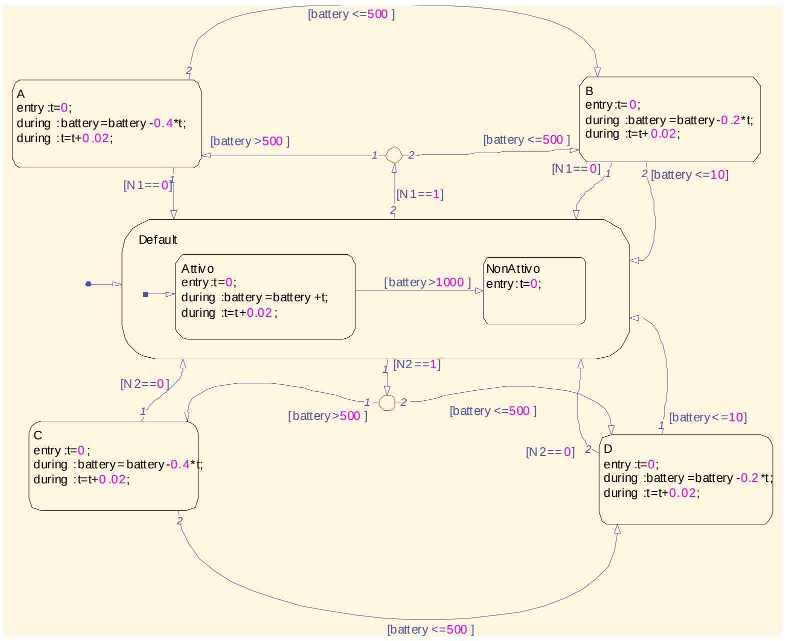


Figure 6.4 Hybrid automata Matlab Prototype

6.1.2.2 Operating conditions approach with MPC Control

The second prototype aims at being more efficient and flexible to cope with the scalability issues that in a complex system may arise. This has been obtained by clustering the representation of the configurations in amore restricted environment: the operating conditions. Given an Embedded System (pSHIELD Node) it is possible to identify a set con elements (battery, buffers, CPU) that can be associated to an operating conditions: a buffer can be saturated, full or empty; a CPU can be idle, working or overloaded; a battery can be full or empty. All these components can also be broken. The combination and aggregation of these conditions allows to create an exhaustive model of a pSHIELD node, as depicted in Figure 6.5. The aggregation is possible, since some behaviours of the components have the same effect of the system (if the CPU or the Buffer is full, the result is always the impossibility of processing data).

At this point the problem of scalability of composition is solved, since the introduction of a new node in the system doesn't imply an exponential increase in the model size, but a linear growth (6 states for each additional node and 4 states for each additional network layer).

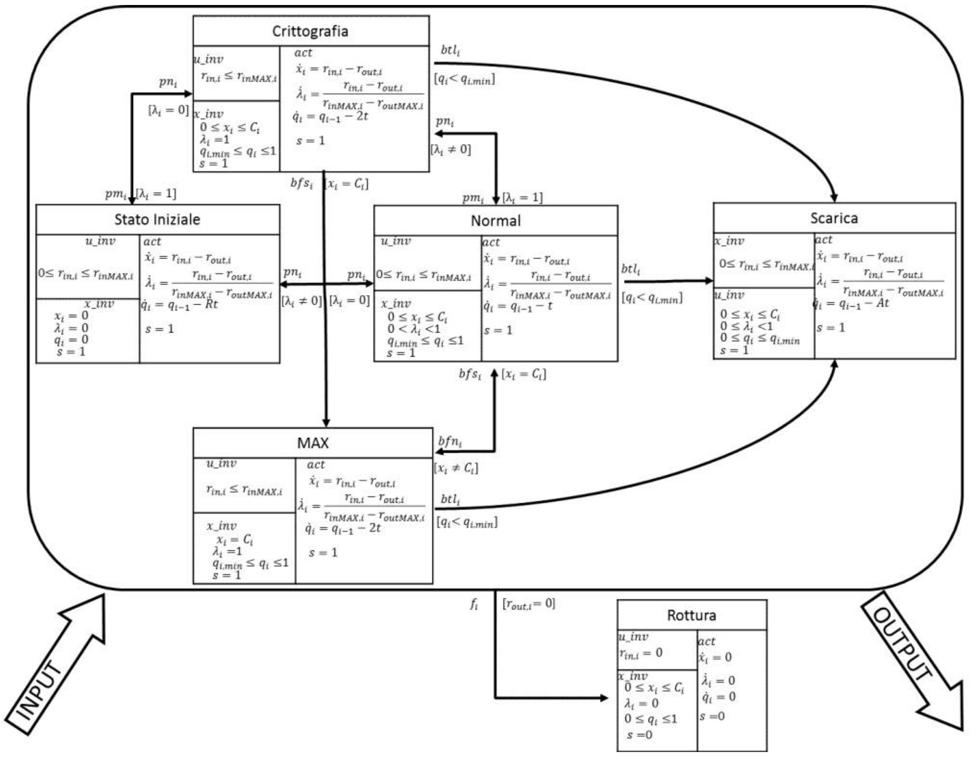


Figure 6.5 Hybrid Automata representing the pSHIELD node

Last, but not least, interesting control algorithms can be applied to the system model due to its formulation by means of these operating conditions (see for example the work of Bemporad [106] and [107]). In particular for the pSHIELD purposes the framework developed in [107], based on Model Predictive Control (MPC), has been considered to verify the effectiveness of the Hybrid Automata approach.

For the simulations it has been used the Matlab Toolbox for Hybrid System with the default configuration (standard MPC problem). The Objective of the control algorithm has been to maximize the amount of data processed by the node while preserving the battery and leaving a certain amount of "reserved" resources for potential emergency tasks.

The assessment of this technology is reported below:

Table 6.2: pSHIELD Composition algorithms: Hybrid Automata approach – ASSESSMENT

pSHIELD Composition algorithms: Hybrid Automata approach - ASSESSMENT	
Limits of this approach to be overcome	The main limits of the hybrid automata approach are: • The scalability: even if the number of states has been reduced, it is still unmanageable in systems composed by thousands of devices (and this value is reasonable). • The expressiveness of the model is limited to simple devices and is not able to capture all the complexity of Embedded Systems world (abstraction is needed) • The difficult implementation of the control laws in a software environment (e.g. Java or C language)
Positive aspects to be preserved	The merging of the Common Criteria approach optimized by a Context Aware control law is a valuable intuition that harmonizes regulations and theory, so it should be preserved.

6.2 nSHIELD potential investigations

6.2.1 Proposed SHIELD Security Agents architecture

[…] This task aims to design and implement an overlay layer based on a system of reacting security agents. The outcome of this task will be a software implementation of a security agent prototype ready to be integrated and interwork with the rest of SHIELD architecture. […]

The SHIELD overlay functionality is implemented through a *security agent* component. This component actually controls a given SHIELD 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).

[…] The security agent will be designed and developed to build autonomously an overlay network composed by different security agents that monitor SPD among groups of embedded system peers, networks, applications and services. Each security agent will interpret the information shared in the SPD system in order to discover imminent threats, menaces and vulnerabilities. All security events of interest will be correlated with the underlying criticality rating the targeted asset. This will results in accurate prioritization and enables fast response to the threats, targeting most critical assets. […]

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.

The development view is depicted in the figure below.

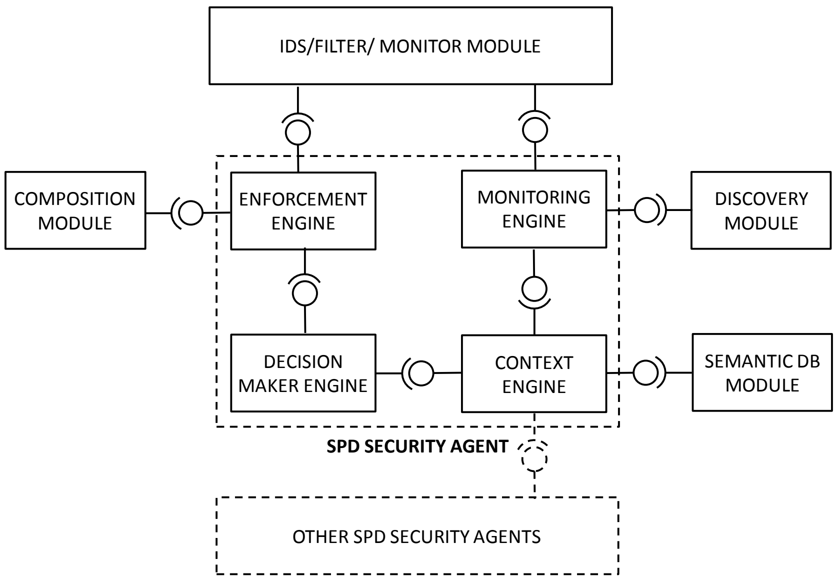


Figure 6.6: nSHIELD SPD Security agent architecture

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

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

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

[…] The security agent reacting system will be a combination of network scanning, passive network monitoring, and integration with existing data provided by the layers. It allows the security agent to organize the network assets into categories. This feature will permit to assign ad-hoc security policies for monitoring each application or service component. […]

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

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

6.2.1.1 Hybrid Agent Systems

[…] A multi-agent approach which combines intelligent, adaptive, autonomous and cooperative capabilities of the agents will be developed. Teams of security agents will cooperate to monitor over time the SPD level on the whole service chain. Therefore, in order to guarantee security and dependability in inter-agent communication, new semantically enriched communication protocols and distributed algorithms capable of dynamically identifying potential dangerous activities, will be analyzed and defined. […].

A security agent is in charge of a SHIELD cluster or subsystem. It is foreseen that security agents can exchange information for control purposes, so a multi-agent approach has to be explored, at least at theoretical level, in the scope of the nSHIELD project. In order to ease the analysis and to keep the solution as much generic as possible, we will focus on a specific class of agents that allows several implementation: the *hybrid agent*s ([108]).

A natural way to design a hybrid agent is to treat the behaviours of the agent as separate subsystems. This style of design leads to the construction of a hybrid agent as a hierarchy of interacting *layers*. In the layered agent architecture we will typically have at least a reactive layer and a proactive layer. However, further layers can be included to equip the agent with additional kinds of behaviour. For example, we may include layers for deduction, communication, social interaction, mobility, adaptation, and other common agent properties. There are two main ways to structure the layers of a hybrid agent, illustrated in Figure 6.7:

1. In a *parallel layered* system, each layer takes the input from the environment separately and produces suggestions as to the necessary output action. In effect, each layer acts as a separate agent.

2. In a *sequential layered* system, the input from the environment is passed through all the layers of the system, and handled by at most one layer. The layers act in concert to ensure that the input is handled appropriately.

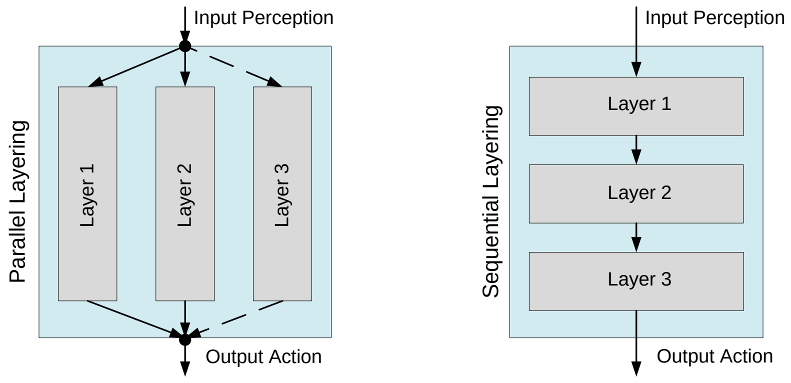


Figure 6.7: Hybrid Agent Architecture

The parallel layered approach has the advantage of conceptual simplicity, as the layers can be implemented independently. In essence, we can add a new layer for each kind of behaviour that we want the agent to have. However, this simplicity is offset by the need to resolve potential conflicts between the layers, and decide which layer has control of the agent at a given time. These tasks are usually assigned to a separate *mediator* that enforces consistency between the layers. The design of this mediator is nontrivial as it may be required to consider all possible interactions between the different layers. Consequently, the mediator may adversely act as a bottleneck inside the agent.

The difficulties of the parallel layered architecture are addressed to an extent by the sequential layered approach. The environmental input flows between all the layers without the need for a mediator. One layer is responsible for the input, and one layer is responsible for the final output action. However, each of the layers must be explicitly designed to fit with the others. A variant of this approach is a *two-pass* architecture, where the input flows up the layers, and the output .flows back down the layers. This variant is analogous to a network protocol stack.

Hybrid agent systems, constructed as layers, are currently the most popular kind of agent architecture. The layered approach is appealing from a pragmatic point of view, as it allows us to define an agent as a composition of different subsystems. These layers can be defined independently, and composed in different ways, affording us considerable flexibility in the design of our agents. For example, we can separate the activities of communication and reasoning, and we can further separate reasoning into reactive and proactive behaviours.

The main criticism of the layered approach is that it is inherently difficult to reason about the behaviour of an agent as a whole. Each layer will typically be defined in a different formalism with its own semantics. Combining these formalisms to provide a unified view of the agent may be a challenging task. Another criticism concerns the interaction between the layers. If the layers are independent, then it is necessary to consider all the ways that the layers can interact in order to reason about the behaviour of an agent.

6.2.2 Proposed SHIELD composition algorithms: DES and Petri Nets

A Discrete Event System (DES) is a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time. In particular a DES is characterized by:

1. The feasible events set *E*;
2. The state space constituted by the discrete set *X*;
3. The state transition mechanism is event-driven.

Formally DES can be modeled by Automata and Petri Nets (PNs). The former is the basic DES model (Cassandras C.G. and Lafortune S. [109], Wonham W.M. [110]), that is characterized by an intuitive structure/graph and easy composition operations, automata are amenable to analysis as well. On other hand, automata model suffers from state space explosion, then automata should be not adequate for complex system. The latter modelling formalism allows the representation of lager class of DES, in fact an automaton can always be represented as a Petri net, but the contrary is not true. Introduced by C. A. Petri in (Petri C.A. [111]) PNs have more structure than automata, although they do not possess, in general, the same analytical power as automata. It is important underline that PN allows to explicit the conditions to enable the event. Furthermore, Petri nets allow to overcome the state space explosion, in particular the Colored Petri Nets (CPNs), an extension to PNs, are developed to model complex systems. CPNs combine the PNs structure with the high-level programming, i.e., using data types and complex data manipulation. Furthermore CPNs allows to obtain a hierarchical descriptions of system, combining a set of sub-models and defining the interfaces between these sub-models.

6.2.2.1 Automata Formal Description

Considering DES, the set E represents the *alphabet* and each (finite) sequence of events is a *word* or *trace*. Then the set of feasible sequences of event that the system can execute is denoted as *language*. To be more precise, a language defined over an event set E is a set of finite-length strings/word formed from events in E. To represent a DES, there are two levels of abstraction: timed or logical. The timed DES state evolution can be expressed by a sequence of the couples event and its occurrence time, i.e., $\{(e_1, t_1), (e_2, t_2)\ldots (e_n, t_n)\}$, whereas logical DES state evolution can be simply expressed by a trace i.e., a sequence of events $\{e_1, e_2, \ldots, e_n\}$. Note that, stochastic DESs are including in the first level of abstraction timed model. We will consider this second level of abstraction where the language models the behaviour of the system. In order to model a system and represent a language, we consider the untimed discrete event modelling formalism *automata*.

A deterministic Automaton is defined in few works as a six-tuple $G = (X, E, f, \text{T}, x_0, X_m)$, where T is the feasible event function defined as T: $X \rightarrow 2^E$, T(x) is the set of events e for which $f(x,e)$ is defined (clearly this second function is derived by transition function f). The automaton can be represented by the state transition diagram, a directed graph to describe graphically the behaviour of system. The state transition diagram nodes represent the states $x \in X$ and the arcs represent the transition labelled by $e \in E$, finally the initial state and marked states are generally identified by an arrow and double circle respectively.

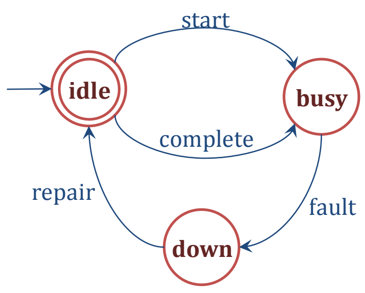


Figure 6.8: state transition diagram of queuing system with breakdowns

For example, the simple automaton in Figure 6.8 represents an elementary queuing (or machine) system with breakdown. The system is defined by the following model:

X = {idle, busy, down};
E = {*start*, *complete*, *fault*, *repair*};
x_0 = idle;
X_m = {idle};
f (idle, *start*) = busy;
f (busy, *complete*) = idle;
f (busy, *fault*) = down;
f (down, *repair*) = idle.

The state transition diagram highlights the connection between automata and languages; in fact every automaton evolution is associated with a word of the event alphabet *E*.

To building the model of complete system built from models of individual system components we need to define the operation parallel composition, that consent to combine two o more automata.

Definition: The *parallel composition* of $G_1=(X_1, E_1, f_1, x_{01}, X_{m1})$ and $G_2=(X_2, E_2, f_2, x_{02}, X_{m2})$ is the automaton $G_1 \,||\, G_2 := (X_1 \times X_2, E_1 \cup E_2, f, (x_{01}, x_{02}), X_{m1} \times X_{m2})$

$$\text{where} \quad f((x_1,x_2),e)=\begin{cases} (f_1(x_1,e), f_2(x_2,e)) & \text{if } e \in \Gamma_1(x_1)\cap\Gamma_2(x_2) \\ (f_1(x_1,e), x_2) & \text{if } e \in \Gamma_1(x_1)\setminus E_2 \\ (x_1, f_2(x_2,e)) & \text{if } e \in \Gamma_2(x_2)\setminus E_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Hence, the events can be common ($E_1 \cap E_2$) or private ($E_2 \setminus E_1$) ∪ ($E_1 \setminus E_2$). In the former case the automata execute the event simultaneously. In the latter, each component can execute the private events whenever possible.

6.2.2.2 Petri Nets Formal Description

Like an automaton, a Petri net is a formalism to describe Discrete Event System behaviour; in particular they represent the DES transition function. A *Petri net* is a five-tuple (P, T, A, w, M_0) where

- $P = \{p_1, p_2, \dots, p_n\}$ is the finite set of places;
- $T = \{t_1, t_2, \dots, t_m\}$ is the finite set of transitions, such that $P \cup T = \varnothing$ $P \cap T = \varnothing$;
- $A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs from places to transitions $((p_i, t_j)$: Input $I(t_j) = p_i$)and from transitions to places $((t_i, p_j)$ Output $O(t_i) = p_j)$;
- $w : A \rightarrow \{1,2,3,\dots\}$ is the weight function on the arcs;
- $M_0 : P \rightarrow \{0,1,2,3,\dots\}$ is the initial marking;

A Petri Net is a directed, weighted, bipartite graph (with two type of node: place and transition) associated with initial marking M_0. Marking assign a non-negative integer k to each place $p \in P$, in other words every place p_i is marked with k_i tokens, $M(p_i) = k_i$, with $k_i \in [0, \infty)$ and i =1, 2, …, n. Given a PN the number and the distribution of tokens control the transitions execution. In fact a transition t_i is enabled if and only if the number of the tokens in each input place p_j is larger than the weight of arc $w(p_j, t_i)$: t_i is enabled iff $M(p_j) \geq w(p_j, t_i)$ where $p_j = I(t_i)$.

An enabled transition can fire only when the associated event occurs. The firing an enable transition t_i removes $w(p_j, t_i)$ tokens from each input place p_j ($p_j = I(t_i)$) and put $w(t_i, p_h)$ tokens in each output place p_h ($p_h = O(t_i)$).

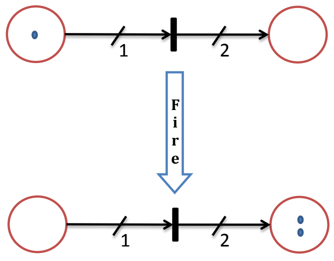


Figure 6.9: example of enabled transition

Petri Nets allow to model the typical features of dynamic systems, such as, concurrency, sequential behaviour, synchronization, mutual exclusive, and so on, in the Figure 6.10 some examples are shown.

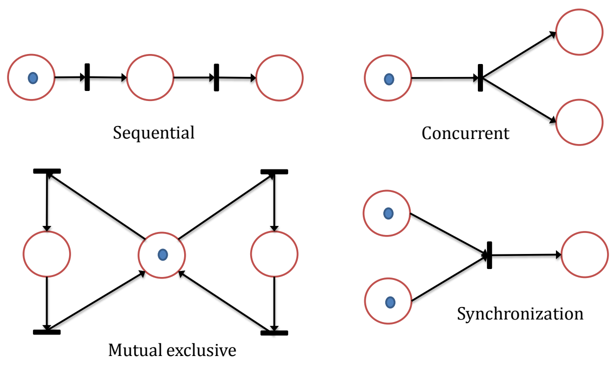


Figure 6.10: Examples of Petri Net primitives

The strength of Petri nets is that they allow to analyze several proprieties associated with the dynamic systems:

- *Reachability*: A marking $M_{n,}$ that represents a specific state of system, is reachable from initial marking M_0 ($M_n \in R(M_0)$) if there exist a sequence of firing $\sigma(M_0) = \{M_0, M_1, \ldots, M_n\}$ that transforms M_0 to M_n.
- *Boundedness*: A PN is k-bounded if the tokens number in each place does not exceed a finite number k for any marking M reachable from initial marking M_0, i.e., $M(p) \leq k$ for every $p \in P$ and every $M \in R(M_0)$.
- *Liveness* (Commoner F. 1972): A transition t is said to be:
  - *Dead* or *L0-live*, if t can never fire in any sequence of firing $\sigma(M_0)$;
  - *L1-live*, if there is some firing sequence $\sigma(M_0)$ such that the transition t can fire at least once;
  - *L2-live*, if the transition t can fire at least k times for some given positive integer k;
  - *L3-live*, if there exists some infinite firing sequence $\sigma(M_0)$ in which the transition t appears infinitely often;
  - *Live* or *L4-live*, if the transition t is L1-live for every possible state reached from M_0.

6.2.2.3 Colored Petri Nets

Kurt Jensen introduced the Colored Petri Nets (CPNs) in 1981; CPNs combine the Petri Nets capabilities with high level programming capabilities. In CPN the tokens carries a data value, referred as token colour, furthermore each place may contain a determinate data type, in other word each place has an associated coloured set. Similarly to PN the marking (number, colours and distribution of tokens) represent the state of the modelled system. The definition of non-hierarchical Colored Petri Net follows:

A *non-hierarchical Colored Petri Net* is a nine-tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, where:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places.
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions such that $P \cup T = \varnothing$ $P \cap T = \varnothing$;
- $A \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs;
- Σ is a finite set of non-empty color sets;
- V is a finite set of typed variables such that Type(v) $\in \Sigma$ for all variables $v \in V$;
- $C : P \rightarrow \Sigma$ is a color set function that assigns a color set to each place;
- $G : T \rightarrow$ exprV is a guard function that assigns a guard to each transition t such that Type[$G(t)$] = Bool;
- $E : A \rightarrow$ exprV is an arc expression function that assigns an arc expression to each arc a such that Type[$E(a)$] = $C(p)$, where p is the place connected to the arc a;

- $I : P \rightarrow expr\varnothing$ is an initialization function that assigns an initialization expression to each place *p*.

The strength of CPNs is the hierarchical structure, composed by modules, that allow to model complex and to work at different abstraction level. The definition of Colored Petri Net Module follows:

A *Colored Petri Net Module* is a four-tuple CPNM = (CPN, T_{sub}, P_{port}, P_{type}), where:

- $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ is a non-hierarchical Colored Petri Net;
- $T_{sub} \subseteq T$ is a set of substitution transitions;
- $P_{port} \subseteq P$ is a set of port places;
- $P_{type} : P_{port} \rightarrow \{IN, OUT, I/O\}$ is a port type function that assigns a port type to each port place.

We refer to [112], [113] and [114] for detailed description of the concepts, analysis methods and practical use of colored Petri nets.

6.2.2.4 Modelling abstract SPD functionality

The DES theory described so far will be used to model the abstract component defined in the semantic activities. An example of the representation of atomic SPD functionalities that exploit the dependable monitoring of goods (see pilot project demonstration), as used by the control engine to drive the composition, is reported in Figure 6.11, where all the options (with their associated level of SPD) are listed, ready to be composed according to the user needs.

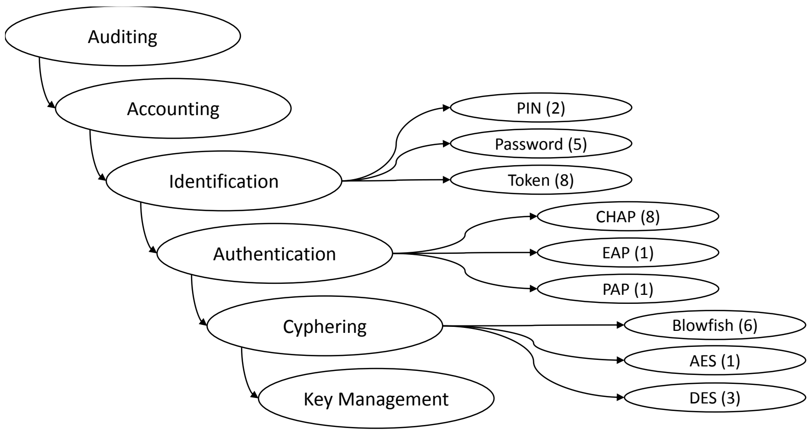


Figure 6.11: SPD functionalities composed for the demonstrator

Considering this simple scenario it is possible to highlight the main features of Automata model: easy composition but scalability problem. The Figure 6.12 shows the simple model of two element of system, assuming that each functionality is always active and can be implemented in three different ways. The state represent in which way is implemented every functionality, the transitions occurs when the functionality change mode and the output is the value of associated level of SPD.

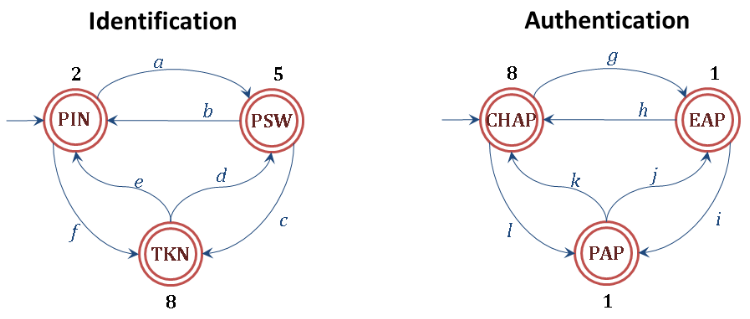


Figure 6.12: SPD functionality Automata model

The basic composition of two functionalities is shown in Figure 6.13. Clearly, the main issue of this approach is the state space explosion.

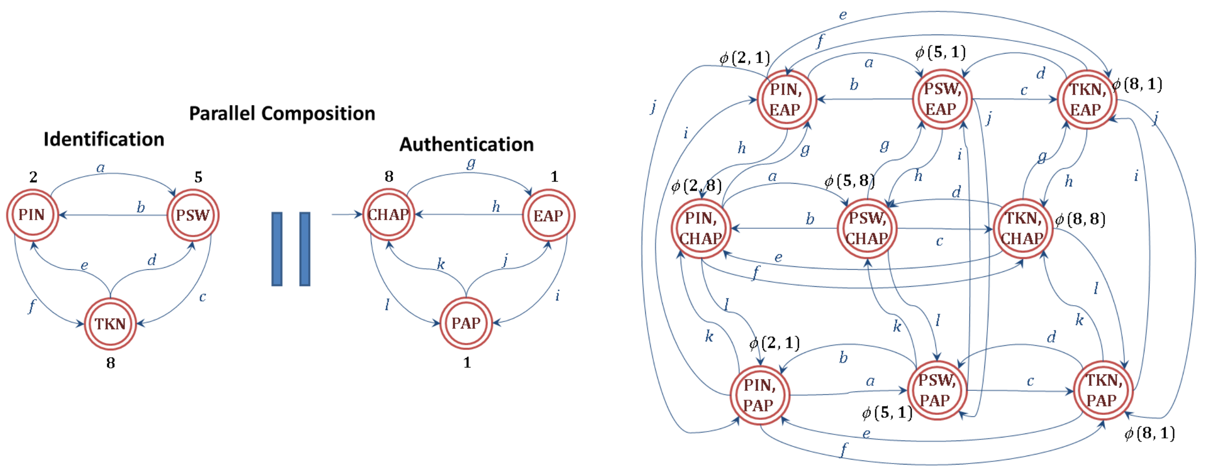


Figure 6.13: SPD functionalities parallel composition

The Petri net model is shown in Figure 6.14, the number of the token represent the associated level of SPD. The control is not performed in fact the transitions are always active.

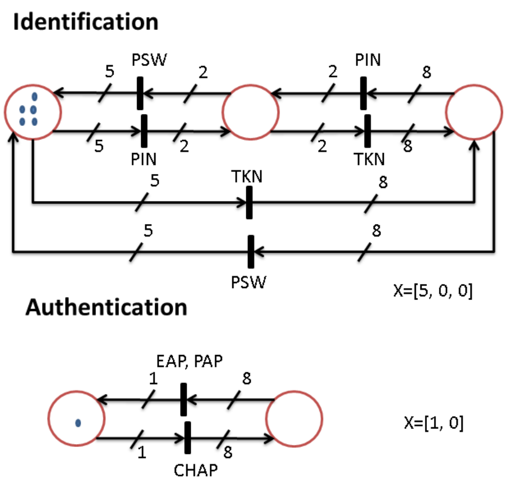


Figure 6.14: SPD functionalities Petri Nets

The composition of system elements is shown in Figure 6.15. Cleary, we are considering a simple system but the difference between automata and PN model are shown. The PNs and, in particular, the CPNs allow to overcome automata issues.

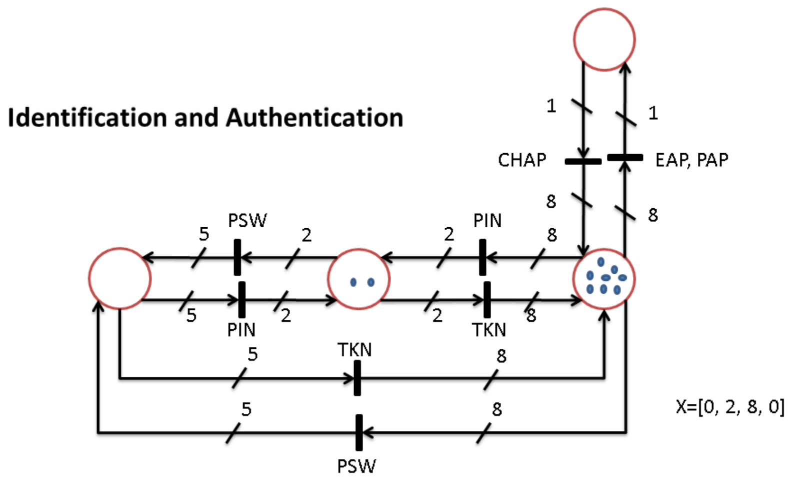


Figure 6.15: Petri nets composition

Finally the main strengths of CPNs are the token color and the sub-models structure. The former is a data value attached to each token; this can be examined and modified by the occurring transitions. The latter allows to overcome scalability problem using a hierarchical description of system.

A reasonable way to model the system is consider multiple cooperating sub-modules which achieve a complete system. The SPD functionalities CPN model, shown in Figure 6.16 and Figure 6.17, is our basic sub-module.

COLOR SET DEFINITION	
Color **SPDname**	String
Color **SPDvalue**	Integer
Color **SPD**	Product **SPDname*SPDvalue**

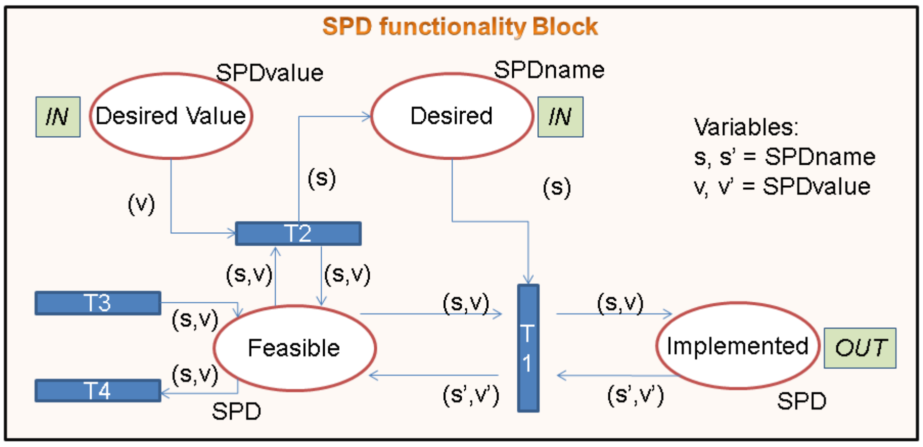


Figure 6.16: SPD Functionalities CPN model

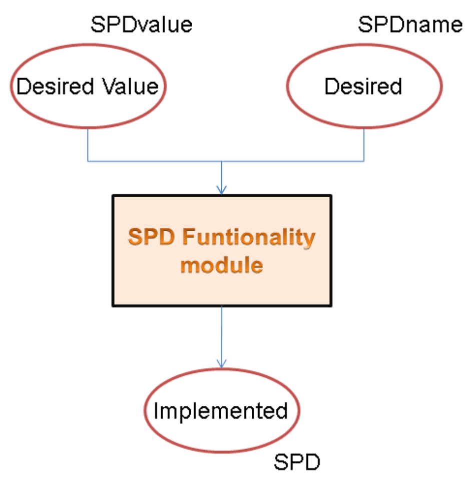


Figure 6.17: SPD Functionality module

This model advantage is the hierarchical structure, that allows to overcome scalability problems and to model heterogeneous systems using the composition of simple basic elements representing SPD functionalities.

7 Conclusions

In this document the assessment of the pSHIELD and nSHIELD technologies have been performed. In particular, for each technology investigated in the pilot phase, an assessment table has been provided to highlight limits and benefits of the approach. Then, this table is used as input to define the new research that has to be performed in the second phase.

The Technical Annex has been reported as well, in order to demonstrate the rationale behind WP5 activities and the strategy to build the SHIELD middleware that is the main enabling technology of the SHIELD framework.

The technologies identified for the prosecution of the nSHIELD project are in a nutshell:

- A new procedure to derive the SHIELD ontology, based on decoupling between abstraction and domain
- A new lightweight language to translate the metamodels
- The secure discovery protocol
- The trusted composition procedure
- The choreographer as manager of composability
- The intrusion detection monitoring and filtering service
- The adapters for legacy systems
- The SHIELD Security Agent
- The multi-agent interaction between security agents
- The policy based management architecture
- The policy definition
- The DES and Petri Nets theory for composability
- The protection profile of SHIELD middleware

The advances will be shown in D5.2, D5.3, D5.4 and D5.5.

8 References

[1] Organization for the Advancement of Structured Information Standards, OASIS, http://www.oasis-open.org

[2] The Web Services Security: SOAP Message Security, WS-Security, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[3] SAML Token Profile 1.1, http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf

[4] Kerberos Token Profile 1.1, http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf

[5] X.509 Token Profile 1.1, http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf

[6] Rights Expression Language (REL) Token Profile 1.1, http://www.oasis-open.org/committees/download.php/16687/oasis-wss-rel-token-profile-1.1.pdf

[7] Username Token Profile 1.1, http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf

[8] Francois Lascelles, Aaron Flint: WS Security Performance. Secure Conversation versus the X509 Profile, 2006, http://websphere.sys-con.com/node/204424

[9] http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.pdf

[10] Web Services Secure Conversation v1.4, WS-SecureConversation, http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.pdf

[11] Hongbin Liu, Shrideep Pallickara, Geoffrey Fox: Performance of Web Services Security, 2005.

[12] Web Services Security Policy v1.3, WS-SecurityPolicy, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.pdf

[13] Web Services Policy 1.5 – Framework, http://www.w3.org/TR/ws-policy/

[14] Web Services Dynamic Discovery v1.1, WS-Discovery, http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.pdf

[15] Web Services Federation Language v1.2, WS-Federation, http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.pdf

[16] Devices Profile for Web Services v1.1, DPWS, http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[17] Zeeb, E., Moritz, G., Timmermann, D., & Golatowski, F. (2010). WS4D: Toolkits for Networked Embedded Systems Based on the Devices Profile for Web Services. 2010 39th International Conference on Parallel Processing Workshops (pp. 1-8). IEEE.

[18] Service Infrastructure for Real time Embedded Networked Applications , ITEA SIRENA,2003-2005, http://www.sirena-itea.org/

[19] Service-Oriented Device & Delivery Architecture, SODA, http://www.soda-itea.org/

[20] Service-Oriented Cross-layer Infrastructure for Distributed Smart Embedded Devices”, SOCRADES, http://www.socrades.eu/Home/default.html

[21] Service-Oriented Architecture for Devices , SOA4D, http://cms.soa4d.org/

[22] Web Services for Devices, WS4D, http://ws4d.e-technik.uni-rostock.de/

[23] Constrained Application Protocol, CoAP , https://datatracker.ietf.org/doc/draft-ietf-core-coap/

[24] IPv6 over Low power Wireless Personal Area Networks, 6LoWPAN, http://tools.ietf.org/wg/6lowpan/

[25] Network-centric Middleware for group communications and resource sharing across heterogeneous embedded systems, MORE, http://www.ist-more.org/

[26] C. Timm, J. Schmutzler, P. Marwedel, C. Wietfeld: "Dynamic Web Service Orchestration applied to the Device Profile for Web Services in Hierarchical Networks", ICST/IEEE 4th International Conference on Communication System Softwareand Middleware, 16th - 19th June 2009, Trinity College Dublin, Ireland.

[27] J. Schmutzler, S. Rohde, C. Wietfeld: "Integration of Wireless Peer-to-Peer Sensor Networks with Embedded Web Services", 14. ITG Fachtagung -Mobilkommunikation, 13th and 14th May 2009, Osnabrück, Germany.

[28] T. Stavropoulos, D. Vrakas, I. Vlahavas, “A Survey of Service Composition in Ambient Intelligence Environments”, Artificial Intelligence Review, Springer, 2011.

[29] Homa Movahednejad, Suhaimi Bin Ibrahim, Mahdi Sharifi, Harihodin Bin Selamat, and Sayed Gholam Hassan Tabatabaei. 2011. Security-aware web service composition approaches: state-of-the-art. In Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS '11). ACM, New York, NY, USA, 112-121.

[30] N. Ibrahim and F.L. Mouel, " A Survey on Service Composition Middleware in Pervasive Environments",International Journal of Computer Science Issues, Volume 1, pp1-12, August 2009.

[31] Shanshan Jiang, Yuan Xue, and Douglas C. Schmidt. 2009. Minimum disruption service composition and recovery in mobile ad hoc networks. Comput. Netw. 53, 10 (July 2009), 1649-1665.

[32] Brent Lagesse, Mohan Kumar, Matthew Wright. ReSCo: A middleware component for Reliable Service Composition in pervasive systems. In Eigth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2010, March 29 - April 2, 2010, Mannheim, Germany, Workshop Proceedings. Pages 486-491, IEEE, 2010.

[33] Jong-Hyun Park and Ji-Hoon Kang. 2011. Intelligent service processing in common USN middleware. Artif. Intell. Rev. 35, 1 (January 2011), 37-51.

[34] Sheikh I. Ahamed, Moushumi Sharmin, A trust-based secure service discovery (TSSD) model for pervasive computing, Computer Communications, Volume 31, Issue 18, 18 December 2008, Pages 4281-4293.

[35] M.J. Kim, M. Kumar, B.A. Shirazi, Service discovery using volunteer nodes in heterogeneous pervasive computing environments, Pervasive and Mobile Computing, Volume 2, Issue 3, September 2006, Pages 313-343.

[36] Fang Shen, Qingqi Pei, Shu-po Bu, A Trust-based Dynamic Secure Service Discovery Model for Pervasive Computing, 2011 Seventh International Conference on Computational Intelligence and Security (2011), Volume: 31, Issue: 18, Publisher: IEEE, Pages: 630-634.

[37] Rafael Moreno-Vozmediano, A hybrid mechanism for resource/service discovery in ad-hoc grids, Future Generation Computer Systems, Volume 25, Issue 7, July 2009, Pages 717-727.

[38] Haitham Elwahsh, Mohamed Hashem, Mohamed Amin, Secure Service Discovery Protocols for Ad Hoc Networks, Communications in Computer and Information Science, 2011, Volume 131, Part 1, 147-157.

[39] Eduardo Moschetta, Rodolfo S. Antunes, Marinho P. Barcellos, Flexible and secure service discovery in ubiquitous computing, Journal of Network and Computer Applications, Volume 33, Issue 2, March 2010, Pages 128-140.

[40] OASIS Devices Profile for Web Services (DPWS) Version 1.1, available on line: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[41] Web Services architecture, available on line: http://www.w3.org/TR/ws-arch/

[42] Geoff Mulligan. 2007. The 6LoWPAN architecture. In Proceedings of the 4th workshop on Embedded networked sensors (EmNets '07). ACM, New York, NY, USA

[43] WS4D-Gsoap, available on-line http://ws4d.e-technik.uni-rostock.de/gsoap/

[44] V. Mareeswari, Dr. E. Sathiyamoorthy: A survey on Trust in Semantic Web Services. International Journla of Scientific & Engineering Research, Volume 3, Issue 2, February. (2012)

[45] Hien Trang Nguyen, Weiliang Zhao, Jian Yang: A Trust and Reputation Model Based on Bayesian Network for Web Services. 2010 IEEE International Conference on Web Services. (2010)

[46] Xing Su, Minjie Zhang, Yi Mu, Kwang Mong Sim: PBTrust: A Priority-Based Trust Model for Service Selection in General Service-Oriented Environments. 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing. (2010)

[47] Surya Nepal, Wanita Sherchan, Athman Bouguettaya: A Behaviour-Based Trust Model for Service Web. IEEE International Conference on Service Oriented Computing and Applications, (2010)

[48] Ming Qu, Shufen Liu, Tie Bao: On theTrusted Ontology Model for Evaluating the Semantic web Services. 14th International Conference on Computer Supported Cooperative Work in Design, (2010)

[49] S. Park, L. Liu, C. Pu, M. Srivatsa, J. Zhang: Resilient Trust Management for Web Service Integration. IEEE International Conference on Web Services. (2005)

[50] Shahab Mokarizadeh, Nima Dokoohaki, Mihhail Matskin, Peep Kungas: Trust and Privacy Enabled Service Composition using Social Experience. 10th IIFIP International Conference on e-business, e-services and e-society (2010), Buenos Aires, Argentina (3-5 Nov). (2010)

[51] Li Qilong, Xin Mingjun, Li Weimin, Zhang Rui: A Trusted Composition Evaluation Model to Support Web Services Coordination in Multi Domains. Research Journal of Applied Sciences, Engineering and Technology 4(6): 587-590. (2012)

[52] Yannick Chevalier, Mohamed Anis Mekki, Michael Rusinowitch: Orchestration under security constraints. Formal Aspects of Security and Trust, FAST'09 workshop. (2009)

[53] A. Andrieux, K. Czajkowski, A. Dan, et al, Web Services Agreement Specification (WS-Agreement), March 14 2007, available at: http://www.ogf.org/documents/GFD.107.pdf

[54] H. Ludwig, A. Keller, A. Dan, et al, Web Service Level Agreement (WSLA) Language Specification, January 28 2003, available at: http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[55] OASIS Devices Profile for Web Services (DPWS) Version 1.1, available on line: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[56] Web Services architecture, available on line: http://www.w3.org/TR/ws-arch/

[57] Geoff Mulligan. 2007. The 6LoWPAN architecture. In Proceedings of the 4th workshop on Embedded networked sensors (EmNets '07). ACM, New York, NY, USA

[58] WS4D-Gsoap, available on-line http://ws4d.e-technik.uni-rostock.de/gsoap/

[59] T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, available on-line: http://tools.ietf.org/html/rfc5246

[60] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", available-online: http://www.ietf.org/rfc/rfc2617.txt

[61] Web Services Dynamic Discovery (WS-Discovery) Version 1.1, available-online: http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html

[62] XML Signatures and processing, http://www.w3.org/TR/xmldsig-core/

[63] http://www.microsoft.com/technet/security/bulletin/ms09-063.mspx

[64] 'Immense' network assault takes down Yahoo February 8, 2000
http://www.cnn.com/2000/TECH/computing/02/08/yahoo.assault.idg/index.html

[65] Cyber-attacks batter Web heavyweights Strikes on eBay, Amazon, CNN.com follow Monday Yahoo! Attack, February 9, 2000,
http://www.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/index.html

[66] Internet quiet after three straight days of attacks Strikes hit E*Trade, ZDNet, eBay, Amazon, others
February 10, 2000
http://www.cnn.com/2000/TECH/computing/02/10/denial.attack.01/index.html

[67] See https://www.owasp.org/index.php/XSS

[68] See http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0449

[69] See http://isecpartners.com/files/XMLDSIG_Command_Injection.pdf

[70] See http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

[71] See http://clawslab.nds.rub.de/wiki/index.php/Coercive_Parsing

[72] See http://clawslab.nds.rub.de/wiki/index.php/Soap_Array_Attack

[73] See http://clawslab.nds.rub.de/wiki/index.php/WS-Addressing_spoofing

[74] See http://www.isecpartners.com/files/isec_hill_attackingxmlsecurity_handout.pdf

[75] Oehlert P.: Violating Assumptions with Fuzzing. IEEE Security & Privacy vol 3, issue 2, pp. 58-62. doi: 10.1109/MSP.2005.55 (2005)

[76] Hung, S.-S., & Shing-Min Liu, D. (2008). A user-oriented ontology-based approach for network intrusion detection. Computer Standards & Interfaces, 30(1-2), 78–88. doi:10.1016/j.csi.2007.07.008

[77] J. Undercoffer, J. Pinkston, A. Joshi, T. Finin, ATarget-Centric ontology for intrusion detection, IJCAI Workshop on Ontologies and Distributed Systems, IJCAI'03, August, 2003.

[78] J. Undercoffer, A. Joshi, T. Finin, J. Pinkston, A target centric ontology for intrusion detection: using DAML+OIL to classify intrusive behaviors, To appear Knowledge Engineering Review—Special Issue on Ontologies for Distributed Systems, Cambridge University Press, 2004.

[79] Abdoli, F., & Kahani, M. (2009). Ontology-based Distributed Intrusion Detection System. Development, 65–70.

[80] Isaza, G. A., Castillo, A. G., & Duque, N. D. (n.d.). An Intrusion Detection and Prevention Model Based on Intelligent Multi-Agent Systems , Signatures and Reaction Rules Ontologies, 237–245.

[81] Isaza, G.; Castillo, A.; López, M.; Castillo, L. & López, M. Intrusion Correlation Using Ontologies and Multi-agent Systems. Information Security and Assurance, Springer Berlin Heidelberg, 2010, 76, 51-63.

[82] Corcho, O., López, M., Gómez-Pérez, A., López-Cima, A.: Building Legal Ontologies with METHONTOLOGY and WebODE. In: Benjamins, V.R., Casanovas, P., Breuker, J., Gangemi, A. (eds.) Law and the Semantic Web. LNCS (LNAI), vol. 3369, pp. 142–157. Springer, Heidelberg (2005)

[83] R. Bruni, I. Lanese, U. Montanari, “A basic algebra of stateless connectors,” Theoretical Computer Science, n. 366, pp. 98-120, 2006.

[84] J. A. Goguen, “Categorical foundations for general systems theory,” Advances in Cybernetics and Systems Research, Transcripta Books, 1973, pp. 121–130.

[85] C.A.R. Hoare, “CSP - Communicating Sequential Processes,” International Series in Computer Science, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[86] R. Milner, “A Calculus of Communicating Systems,” Lecture Notes in Computer Science, Vol. 92, Springer, Berlin, 1989.

[87] “Compositional Algebra of CONNECTors”, European Union FP7 ICT CONNECT Project, Tech. Rep. Deliverable 2.2, 2011. Available: http://www.connect-forever.eu

[88] S. Bliudze, J. Sifakis, "Causal semantics for the algebra of connectors," Formal Methods in System Design, Springer Science, n. 36, pp. 167-194, 2010.

[89] J. Zhang, V. Varadharajan, "Wireless sensor network key management survey and taxonomy", Journal of Network and Computer Applications, Elsevier, Volume 33, Issue 2, March 2010, pp. 63–75.

[90] A. Abd EL-Aziz, A.Kannan, "Access Control for Healthcare Data Using Extended XACML-SRBAC Model", 2012 International Conference on Computer Communication and Informatics (ICCCI), pp. 1–4, 2012.

[91] C. Karlof, N. Sastry, D. Wagner, "TinySec: A link layer security architecture for wireless sensor networks", SenSys '04 Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM, pp. 162–175, 2004.

[92] W. Jung, S. Hong, M. Ha, Y.-J. Kim, D. Kim, "SSL-based Lightweight Security of IP-based Wireless Sensor Networks", 2009 International Conference on Advanced Information Networking and Applications Workshops, pp. 1112–1117, 2009.

[93] S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifi, "Tiny 3-TLS: A Trust Delegation Protocol for Wireless Sensor Networks", Security and Privacy in Ad-Hoc and Sensor Networks, LNCS 4357, Springer Berlin / Heidelberg, pp. 32–42, 2006.

[94] A. Amokrane, Y. Challal, A. Balla, "A Secure Web Service-based Platform for Wireless Sensor Network Management and Interrogation", 2011 Conference on Network and Information Systems Security (SAR-SSI), pp. 1–8, 2011.

[95] OASIS, eXtensible Access Control Markup Language (XACML) Version 3.0, Available from http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf

[96] OASIS, SAML 2.0 Profile for XACML, Version 2.0. Available from http://docs.oasis-open.org/xacml/3.0/xacml-profile-saml2.0-v2-spec-en.pdf.

[97] OASIS, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Available from http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[98] OASIS, SOAP-over-UDP Version 1.1. Available from http://docs.oasis-open.org/ws-dd/soapoverudp/1.1/os/wsdd-soapoverudp-1.1-spec-os.html

[99] D. Martin, editor. OWL-S 1.1 Release. 2004. http://www.daml.org/services/owl-s/1.1/

[100] D. Martin, editor. OWL-S 1.2 Release. 2008. http://www.ai.sri.com/daml/services/owl-s/1.2/

[101] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction and Composition of SemanticWeb services. Journal of Web Semantics, 1(1):27{46, September 2003.

[102] Lara, R., Roman, D., Polleres, A., Fensel, D.: A Conceptual Comparison of WSMO and OWL-S. In: Proc. of the European Conf. on Web Services. (2004)

[103] Martin, David, et al. OWL-S: Semantic Markup for Web Services. W3C Member Submission: November 22, 2004. http://www.w3.org/Submission/OWL-S/

[104] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. IEEE Intelligent Systems, 16(2):46–53, 2001

[105] Fensel, D., & Bussler, C. (2002). The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications, 1(2)

[106] Bemporad A. and Di Cairano, S., “Optimal Control of Discrete Hybrid Stochastic Automata”, Proceedigns of ACM International Conference on Hybrid Systems: Computation and Control (HSCC05), pp. 151-167, Zurich, Switzerland, 9-11 March, 2005

[107] Bemporad A. Di Cairano S. and Giorgetti N., “Model Predictive Control of Hybrid Systems with Applications to Supply Chain Management”, Proceedings of 49th Convegno Nazionale ANIPLA, Naples, Italy, Nov, 2005

[108] Tapia D., Bajo J., Corchado J., Rodríguez Sara., Manzano J. "Hybrid Agents Based Architecture on Automated Dynamic Environments", in Knowledge-Based Intelligent Information and Engineering Systems, Lecture Notes in Computer Science, Springer Berlin / Heidelberg 2007

[109] C.G. Cassandras, S. Lafortune, Introduction to Discrete Event Systems - SecondEdition, Springer, 2008. ISBN 978-0-387-33332-8. (771+xxiii pages)

[110] W. M. Wonham, Supervisory Control of Discrete-Event Systems. Ece 1636f/1637s 2009-2010, 2010 [online] Available: http://www.control.utoronto.ca/cgi-bin/dldes.cgi

[111] C.A. Petri, Kommunikation mit Automaten. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962, Second Edition:, New York: Griffiss Air Force Base, Technical Report RADC-TR-65--377, Vol.1, 1966, Pages: Suppl. 1, English translation

[112] K. Jensen,, 'An Introduction to the Theoretical Aspects of Coloured Petri nets', in J Bakker, W Roever & G Rozenberg (eds), A Decade of Concurrency Reflections and Perspectivesvol. 803, Lecture Notes in Computer Science, vol. 803, Springer, pp. 230-272, 1994.

[113] K. Jensen, 'A Brief Introduction to Coloured Petri Nets', in E Brinksma (ed.), Tools and Algorithms for the Construction and Analysis of Systemsvol. 1217, Lecture Notes in Computer Science, vol. 1217, Springer, pp. 203-208, 1997.

[114] K. Jensen, L.M. Kristensen, Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, 2009. 384 p.