Project no: 269317

nSHIELD

new embedded Systems arcHItecturE for multi-Layer Dependable solutions

Instrument type: Collaborative Project, JTI-CP-ARTEMIS

Priority name: Embedded Systems

D5.3 Preliminary SPD Middleware and Overlay Technologies Report

Due date of deliverable: M18 – 2013.02.28

Actual submission date: M22 – 2013.06.30

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

Organisation name of lead contractor for this deliverable:

Selex Electronic Systems, SES

Revision [Final]

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

Document Authors and Approvals

Authors		Date	Signature
Name	**Company**		
Andrea Morgagni	Selex ES		
Andrea Fiaschetti	UNIROMA1		
Balázs Berkes	S-LAB		
Baldelli Renato	Selex ES		
Andrea Lanna	UNIROMA1		
Inaki Arenaza	MGEP		
Panagiotis Soufrilas	ATHENA		
Kostas Rantos	TUC		
Kostas Fysarakis	TUC		
Alexandros Papanikolaou	TUC		
Dimitris Geneiatakis	TUC		
Harry Manifavas	TUC		
Nikos Pappas	HAI		
Gaetano Scarano	UNIROMA1		
Roberto Cusani	UNIROMA1		
Martina Panfili	UNIROMA1		
Andrea Lanna	UNIROMA1		
Silvano Mignanti	UNIROMA1		
Vincenzo Suraci	UNIROMA1		
Francesco Delli Priscoli	UNIROMA1		
Reviewed by			
Name	**Company**		
Elisabetta Campaiola	SES		
Approved by			
Name	**Company**		
Cecilia Coveri	SES		

Applicable Documents

ID	Document	Description
AD1	TA	nSHIELD Technical Annex
AD2	D5.1	Deliverable D5.1: SPD Middleware and Overlay Technologies Assessment
AD3	D2.4	Deliverable D2.4: Reference System Architecture Design
AD4	D2.5	Deliverable D2.5: Preliminary SPD Metric Specification

Modification History

Issue	Date	Description
Draft 0.1	14/01/2013	First version of ToC.
Draft 0.2	03/05/2013	All
Final	30/06/2013	Final review

Executive Summary

This document reports the achievement of the first phase of the nSHIELD project with respect to middleware technologies and prototypes.

The document is structured as follows:

1. Introduction: a brief introduction to the document contents
2. Achievements in the formulation of the SHIELD semantic technologies
3. Achievements in the development of the SHIELD core serviced at Middleware level
4. Deals with the progress in the definition of the SHIELD policies
5. Progress in the Overlay control algorithms
6. Brief conclusions
7. Link to the D5.2 prototypes
8. References

Contents

Figures

Tables

Glossary

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

This Page is intentionally left blank

1 Introduction

The nSHIELD project proposes an innovative layered architecture to provide intrinsic security (SPD features and functionalities) to modern Embedded Systems. This is achieved by enriching these systems with innovative functionalities, among which the most important is the “composability”, i.e. the possibility of dynamically enable/disable/configure system’s components in order to achieve a desired end-to-end behaviour.

Since an Embedded System is a mix of HW and SW functionalities, the intermediate layer (Middleware) plays an important role because it acts as a glue to harmonize all the heterogeneous components and, in the scope of the composability, to orchestrate the system behaviour according to the decision taken by the “intelligent” Overlay.

In spite of this, the nSHIELD project, through WP5 activities, aims at defining the architecture and behaviour of the so called **SHIELD Middleware & Overlay** that will be characterized by:

- Common Middleware functionalities
- Innovative SPD oriented Middleware functionalities
- Mechanisms to compose SPD functionalities to satisfy security needs.

This last point is addressed by means of a virtual vertical layer, transversal to all the others that collects system information and elaborates them to drive the SPD composability. This layer is named **SHIELD Overlay** and, even though it is logically separated by the others, on a practical point of view it can be treated like a middleware component.

The various achievement and prototypes derived so far are reported in the following chapter. They will be refined in the second project delivery and will be enriched with new ones.

2 Semantic Technologies

2.1 SHIELD semantic models

In this paragraph the semantic models of the SHIELD system will be presented.

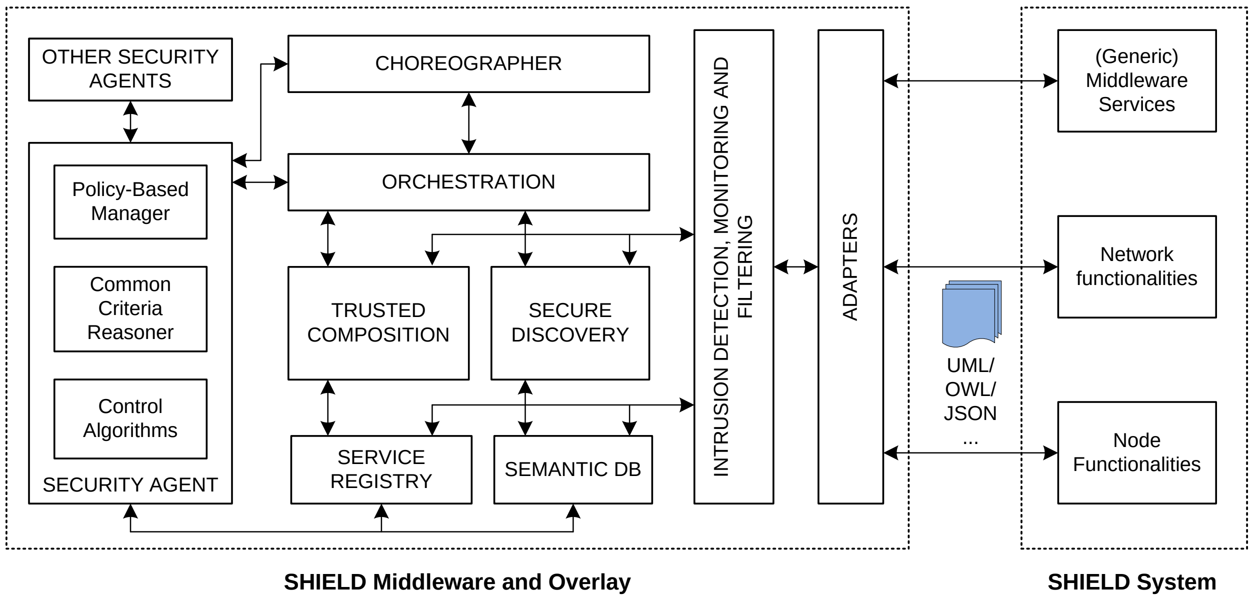


Figure 2-1: Semantic models of the SHIELD system

Today, the evaluation of Security, Privacy and Dependability (SPD) functions is one of major aims of research community and IT industry [1]. Despite all the work done, they do still not reach a convergence solution of Security formal definition. In the same time, the security approach is becoming a fundamental element in every sphere and area. This increasing demand for improvement of security, privacy and dependability has resulted in the increase of products and services to allow “more reliable system”. These new reliable system are typically the results of sub-systems combinations, for example to achieve a security phone, it was added to it and authentication module or a cryptography function.

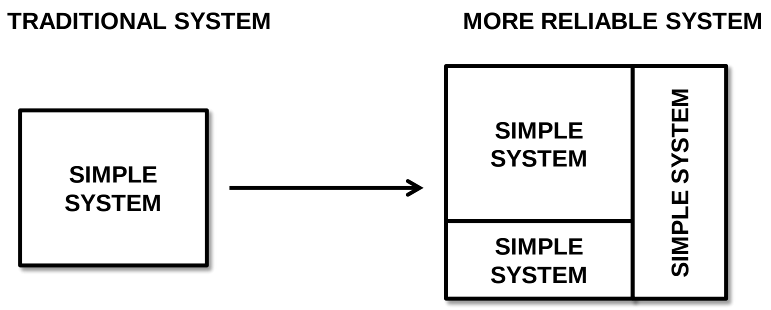


Figure 2-2: Idea of sub-systems compositions

The sub-systems composition implies several interdependences among all those involved. In particular, the state and the functioning of each simple system depend of the states of others.

The major challenge of nSHIELD project in this sphere is to model the SPD context by means of semantic technologies.

To entry in this scenario is necessary an appropriate formalization and, in particular, of “security functionality” concept. An important contribute in this area is given in [2], [3], and [4].

According with those, we use the following definitions:

- **Target of Evaluation (TOE)** is a set of software, firmware, hardware or other possibly thing that is the subject of an evaluation. It may contain resources such as electronic storage media (e.g. main memory, disk space), peripheral devices (e.g. printers), and computing capacity (e.g. CPU time) that can be used for processing and storing information and is the subject of an evaluation.
- **Protection Profile (PP)** that is the implementation-independent statement of security needs for a TOE type.
- **Security Target (ST)** that is the implementation-dependent statement of security needs for a specific identified TOE.

So, the **security functionalities** express security requirements intended to counter threats in the assumed operating environment of the TOE and/or cover any identified organisational security policies and assumptions. Security functional components are the basis for the security functional requirements expressed in a PP or a ST. These requirements describe the desired security behaviour expected of a Target of Evaluation (TOE) and are intended to meet the security objectives as stated in a PP or an ST. These requirements are the **security functionalities** and they describe security properties that users can detect by direct interaction (i.e. inputs, outputs) with the IT or by the IT response to stimulus.

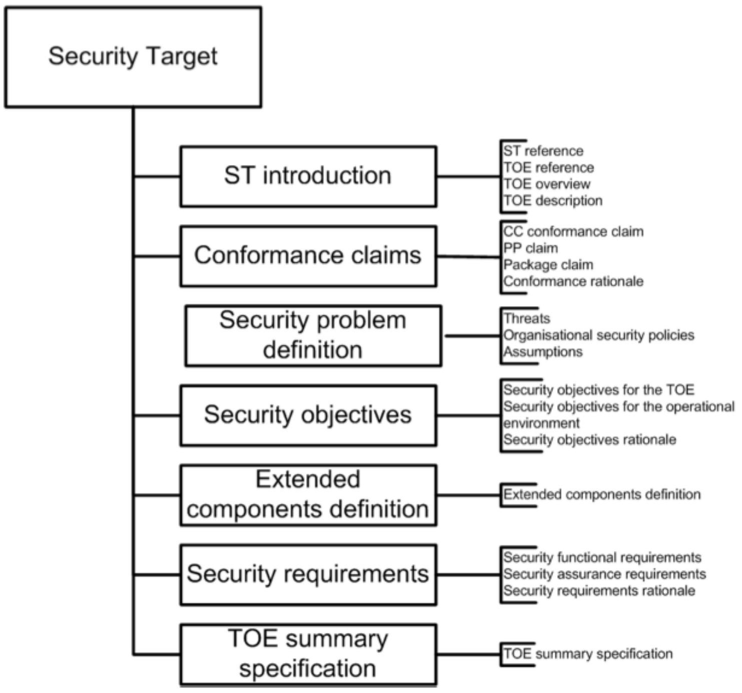

Security Target
ST introduction
ST reference
TOE reference
TOE overview
TOE description
Conformance claims
CC conformance claim
PP claim
Package claim
Conformance rationale
Security problem definition
Threats
Organisational security policies
Assumptions
Security objectives
Security objectives for the TOE
Security objectives for the operational environment
Security objectives rationale
Extended components definition
Extended components definition
Security requirements
Security functional requirements
Security assurance requirements
Security requirements rationale
TOE summary specification
TOE summary specification

Figure 2-3: Security Target contents

In few words **security functionalities** express security requirements intended to counter threats in the assumed operating environment of the TOE and/or cover any identified organisational security policies and assumptions.

Continuing to follow the way suggested in [2], [3], and [4], we dwell on ST and PP formalizations.

Figure 2-3 shows the Security Target context. ST fulfils two roles: to specify “what is to be evaluated” before and during the evaluation and to identify “what was evaluated” after the evaluation. ST is thought to be a security specification on a relatively high level of abstraction and it should not contain detailed component specifications (such as low level protocol details or algorithms disclaimers) or other unless security-relevant facts (e.g. physical weight …).

So, a normally ST contains:

i. *introduction* that contains three TOE descriptions on different levels of abstraction;
ii. *conformance claim*, that shows whether the ST claims conformance to (and which) any PPs;
iii. *security problem definition* with threats and assumptions;
iv. *security objectives*, showing how the security problem solution;
v. *extended components definition* (optional);
vi. *security requirements*, TOE security objectives are translated in a standardized language;
vii. *TOE summary specification* that shows how the security functional requirements are implemented.

It is easy to understand the ST Fields without dwelling again.

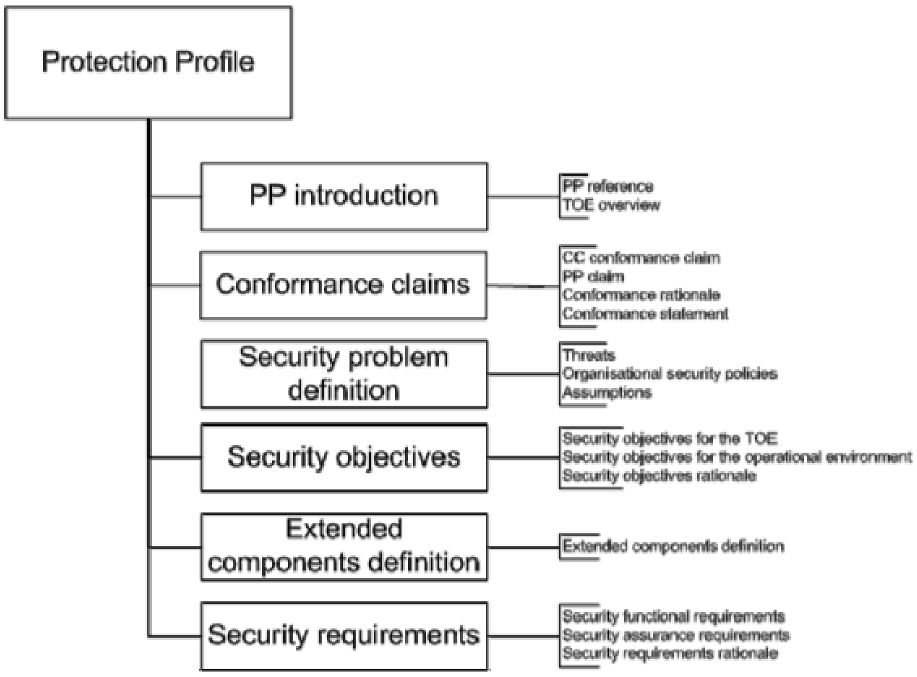

Protection Profile
PP introduction
PP reference
TOE overview
Conformance claims
CC conformance claim
PP claim
Conformance rationale
Conformance statement
Security problem definition
Threats
Organisational security policies
Assumptions
Security objectives
Security objectives for the TOE
Security objectives for the operational environment
Security objectives rationale
Extended components definition
Extended components definition
Security requirements
Security functional requirements
Security assurance requirements
Security requirements rationale

Figure 2-4: Protection Profile context.

Figure 2-4 shows the Protection Profile object. PP is used to provide a requirement specification for a specific consumer or group, to regulate the protection from a specific regulatory entity and to define a baseline for a group of IT developers. As ST object and for the same reasons, PP should not have a too detailed or complete specification, but only the security-relevant ones. Moreover, the PP should not contain single-object specifications because, unlike an ST, a PP is designed to describe a certain type of IT, and not a single product.

The fields of PPs are similar to those of ST, without *TOE summary specification*.

This brief examine shows an interesting approach to describe and manage security functionalities. These contents come from **Common Criteria** (CC) theory that was born:

- to ensure that evaluations of IT products and protection profiles are performed to high and consistent standards and are seen to contribute significantly to confidence in the security of those products and profiles;
- to improve the availability of evaluated, security-enhanced IT products and protection profiles;
- to eliminate the burden of duplicating evaluations of IT products and protection profiles;
- to continuously improve the efficiency and cost-effectiveness of the evaluation and certification/validation process for IT products and protection profiles.

It is clear that the CC has several shared aims with nSHIELD project. We investigated it because we believe that this is an interesting approach and it could help us to achieve the aim of modelling the SPD context by means of semantic technologies. We would underline that CC focuses his effort only on IT systems, instead nSHIELD contemplates a scenario largest. Different scenarios imply several difficulties to develop a unique solution that is always available in each of these. For that, we consider CC only as an interesting approach and not as a solution.

Already in pSHIELD project, and then in nSHIELD, it was been choosing to face the challenge developing a enough generic representation 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 specific application. Likewise, the approach that was been basing on these three guidelines:

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

This concept was explained in Figure 2-5, reducing the modelling problem to the formulation of three different meta-models: i) structure; ii) functions; iii) relations between structure and functions.

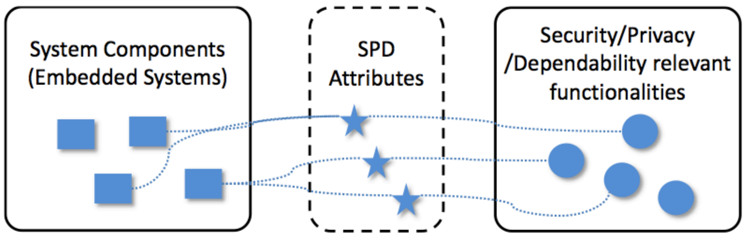

System Components (Embedded Systems)
SPD Attributes
Security/Privacy /Dependability relevant functionalities

Figure 2-5: Proposed approach to model SPD for ES

According with Figure 2-5 and considering the non-scalability of pSHIELD approach, it is necessary provide a relevant abstraction to achieve the SPD system model. In particular, as shown in [5] it was thought to decouple the three pSHIELD metamodels into two new nSHIELD ones:

- a technology independent metamodel;
- a domain library.

This process of decoupling is shown in Figure 2-6, where the SPD abstraction (Technology Independent Abstraction - TIA) is composited by SPD component and functionality, instead the domain dependencies (Domain Dependent Libraries - DDL) include NON-SPD components and functionalities, and the attributes.

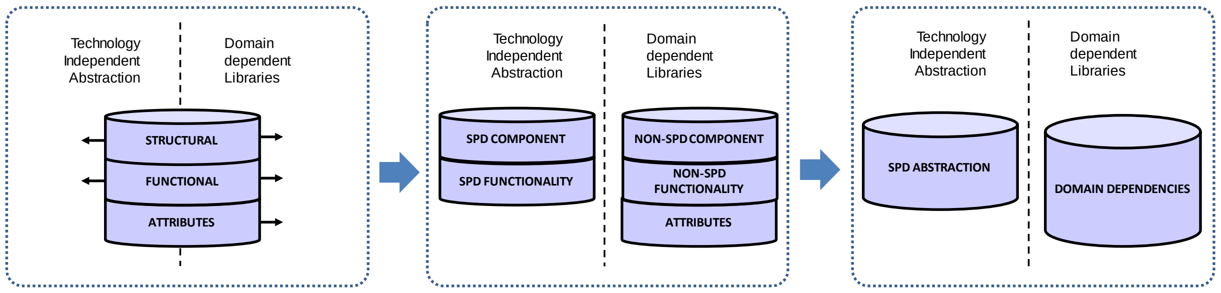


Figure 2-6: Decoupling of the pSHIELD ontology into nSHIELD approach.

To achieve the first metamodels abstraction, a feasible approach is given by CC approach. Using the same formalization and the acronyms to prevent further introduction of terms, the technologies independent abstraction can be represent with three objects:

- TOE, which describes the object that we want SPD-evaluate;
- PP, which includes security aims for type of objects;
- ST, which includes security, aims for single components.

PP and ST also describe the inter-relations among different components or class of these.

The second level of abstraction, between SPD Attributes and SPD Functionalities is given by Domain Library, which associates for each SPD Attributes state a corresponding value of SPD.

Figure 2-7 shows the proposed approach in Figure 2-5 whit these considerations.

We would underline that CC is only the basic idea and we use the same formalizations for easier understanding. This approach does not imply to use the same owner methods, objects or already defined thing.

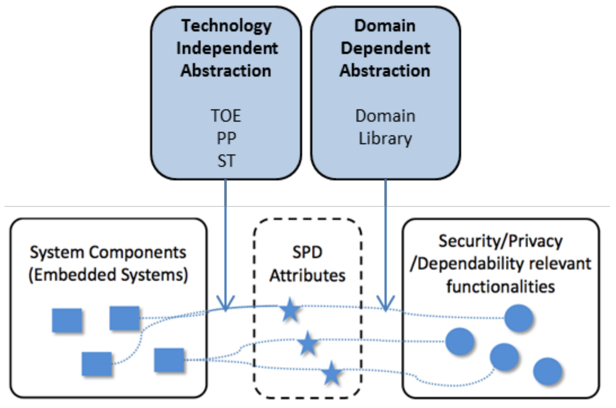

Technology Independent Abstraction
TOE
PP
ST
Domain Dependent Abstraction
Domain Library
System Components (Embedded Systems)
SPD Attributes
Security/Privacy /Dependability relevant functionalities

Figure 2-7: Proposed abstracted approach to model SPD for ES.

This approach also allows to develop a formal verification method, how done in [6]. The authors start from the fact that it is difficult to define reliable security properties that should be applied to validate an IT system and they dwell in particular on satisfaction of the security criteria defined in ISO/IEC 15408.

Nowadays, more organizations and government agencies require the use of CC certified and use the CC methodology in their acquisition, such as U.S. National Information Assurance Acquisition Policy [7].

Therefore, as well as being an interesting approach to follow, a CC conformed method could get important benefit for nSHIELD project.

Re-taking Figure 2-7, the role of TIA is to associate an Abstracted SPD Functionality Model (ASFM) at each system components, considering:

1. components features and structures;
2. interconnections and inferences among components;
3. security policies.

It is clear that for 1 and 2, this abstraction must consider the operative scenario and it is not completely abstracted.

As an example of the new semantic abstraction, we will consider *cyphering* (see Figure 2-8). 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.

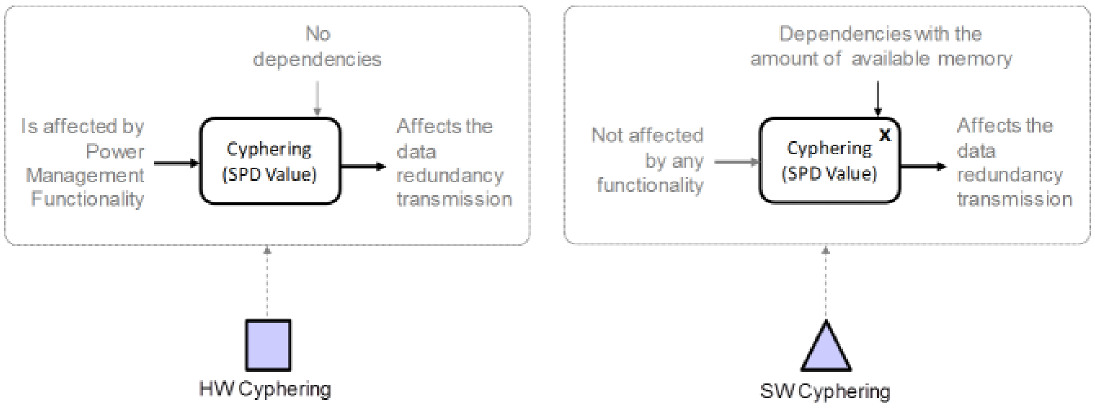

No dependencies
Is affected by Power Management Functionality
Cyphering (SPD Value)
Affects the data redundancy transmission
HW Cyphering
Dependencies with the amount of available memory
Not affected by any functionality
Cyphering (SPD Value)
x
Affects the data redundancy transmission
SW Cyphering

Figure 2-8: 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.

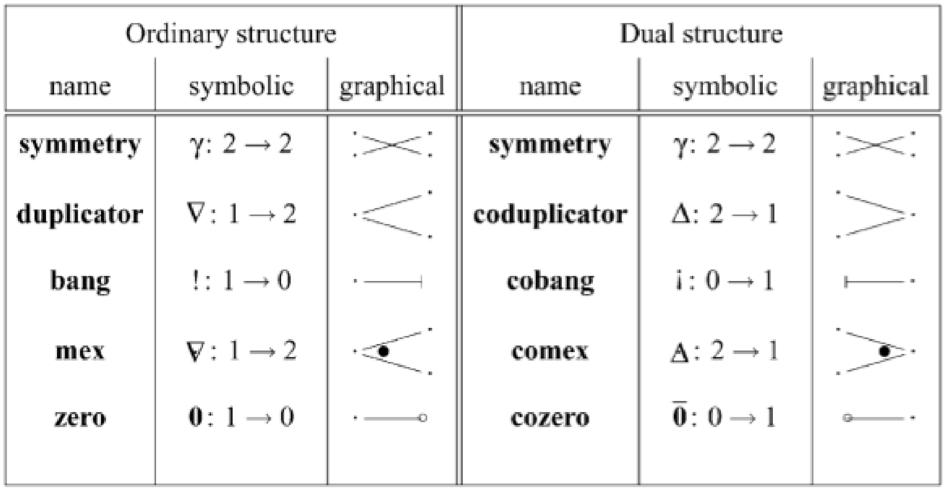
To model SPD attribute and SPD functionalities, we are following **Algebra of Connectors** way: the concept of *connector* has emerged to interact and interoperate small components, functionalities and services that are separately developed. Connectors are the *glue code* that takes care of all those 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" [8]. The connector concept has been developed following two different approach for *system modelling categorical* and for *algebraic approaches* (see [9] [10] and [11] , respectively).

Categorical approach consists of depicting the systems as objects in a category and the sub-system and the relations through appropriate morphisms.

Instead, algebraic approach requires a suitable algebra to model systems. This approach contains only basic components that are used to develop other operators and complex systems.

Several papers [8], [12], and [13] suggest different algebraic approaches always with simple basic components. For example, in [8] the authors present a basic algebra of stateless connectors using only five basic connectors' syntax that is shown in Figure 2-9.



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

Figure 2-9: Example of basic semantic in [8]

Another interesting contribute is given in [13] where the authors define six primitive semantic with different basic elements approach. Figure 2-10 shows the primitive of this algebra.

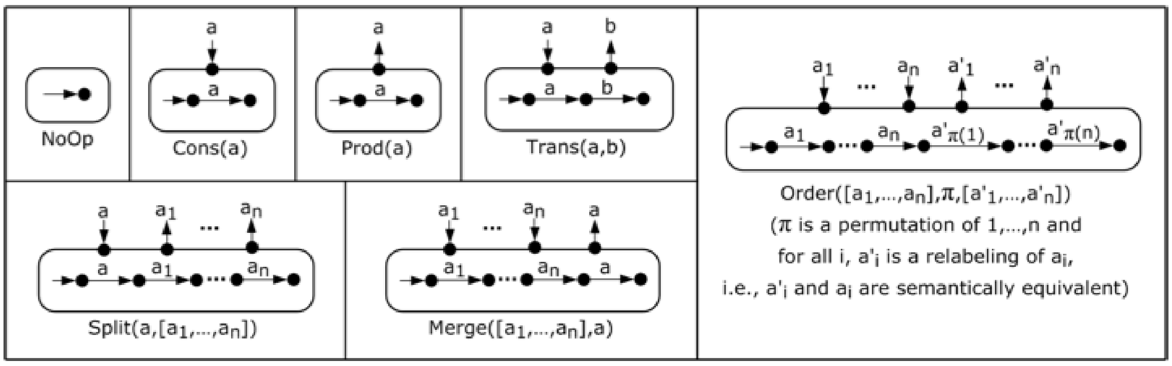

NoOp
Cons(a)
Prod(a)
Trans(a,b)
Order([a1,...,an],π,[a'1,...,a'n])
(π is a permutation of 1,...,n and
for all i, a'i is a relabeling of ai,
i.e., a'i and ai are semantically equivalent)
Split(a,[a1,...,an])
Merge([a1,...,an],a)

Figure 2-10: Example of basic semantic in [13]

Cited documents and other Algebra of Connectors contributions establish that there is not a unique algebra, but it must be developed *ad hoc* considering each single scenario. The system modelling requires a particular attention in order to consider multiple and simultaneous actions.

The main benefit of the development of connector algebra, or at least composition formalism, is to evaluate the individual SPD contribution of each component, which can be used for feedback controlling.

We show Figure 2-9 and Figure 2-10 because there is the most comparable Algebra of Connector that we must develop for Technology Independent and Domain Dependent Abstractions, respectively.

2.1.1 The SHIELD ontology

In spite of all the considerations collected so far, the Ontology model adopted for the SHIELD project is a re-elaboration of the one already presented in pSHIELD, limited to the abstraction layer and including a new set of information directly derived from a new concept, the attack surface metrics, that will be elaborated in the prosecution of the project and that represent the "logical" algebra of connection that interconnects the SPD functionalities.

This approach, that will be better outlined in the Metrics documents in the second phase of the project, is based on the quantification of the so-called "attack surface" and "porosity", that is representative of the vulnerability of the system. This surface is a function of the amount of interfaces to the external world (**access**), interactions between components (**complexity**) and internal/external interactions with no direct impact on security (**trust**). These attributes are represented by a number, so the generic SPD functionality brings a numeric contribution for each of these attributes.

These values hold both for the system and for each additional SPD functionality

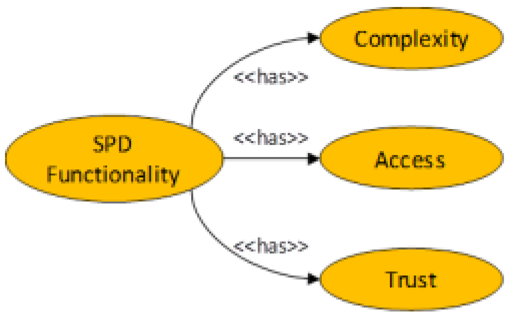


Figure 2-11: SHIELD SPD Functionality Ontology: attributes

Then, each vulnerability (identified by the number of "accesses") can be counteracted by means of specific controls. The controls can be classified in Class A (interactive controls) that are Authentication, Indemnification, Subjugation, Continuity, and Resilience, and Class B (process controls) that are Non-repudiation, Confidentiality, Privacy, Integrity, and Alarm.

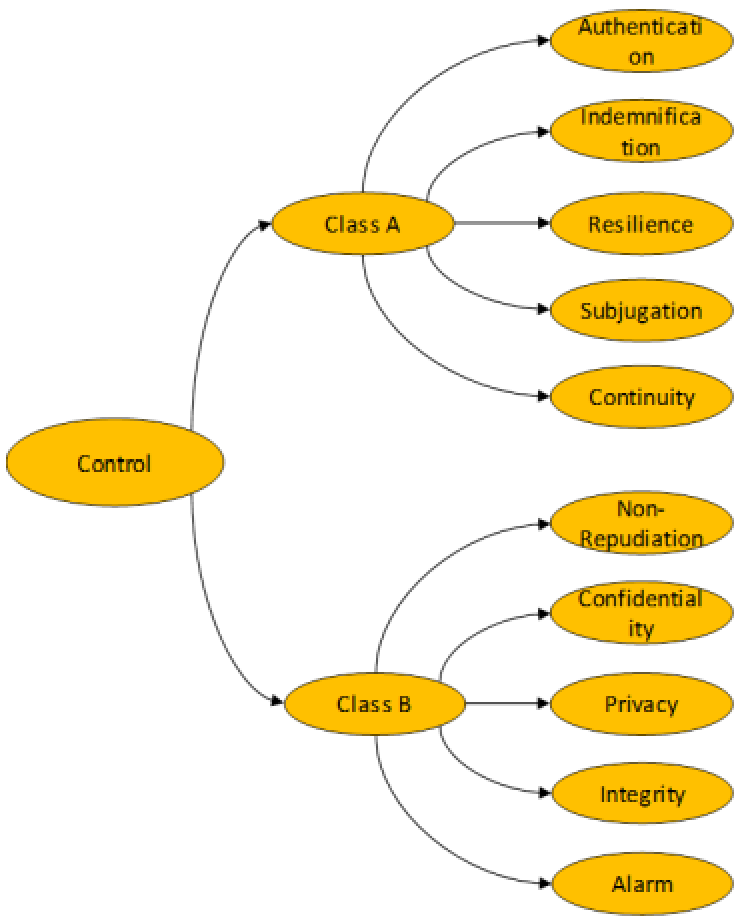


Figure 2-12: SHIELD "Control Ontology"

Each SPD functionality can bring into the system one or more controls. Each control, once activated, can be affected by a set of limitations: Vulnerabilities, Weaknesses, Concerns, Exposures, and Anomalies.

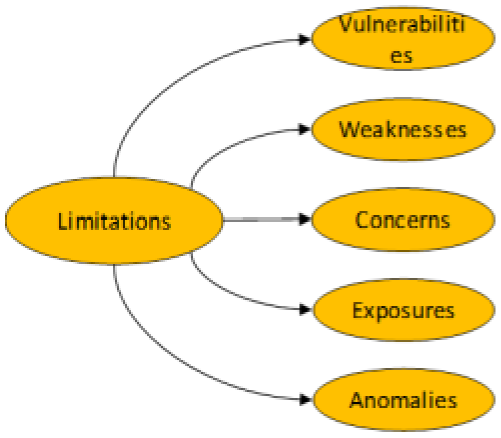


Figure 2-13: SHIELD "Limitations" Ontology

Each element depicted in these Ontologies can be:

- A simple number (i.e. 2 integrity controls, …)
- An element itself (i.e. CRC integrity control, Hash integrity control, …)

These elements are finally composed, according to a proper algebra, to obtain the metric value for the whole system. The algebra will be detailed in the Metrics document.

To sum up, the SHIELD ontology, stored in the SHIELD components, is based on an abstract description of the generic SPD functionality including all the attributes (numerical or instantiated) to compute the contribution given by that functionality to the whole system.

Then a proper domain data base is needed to tailor the result to the specific application scenario.

2.1.2 The SHIELD Domain dependent library

This library contains all the refinements necessary to tailor the abstract components to the specific scenario needs. In particular in contains:

- A list of functional dependencies between the SPD functionalities (mutual inclusion and mutual exclusion)
- A list of numerical values for the metrics attribute of the Ontology previously defined.

This second relation is expressed by duplicating the Ontology tree and by instantiating only the values that override the ones already present in the Ontology. For example if and SPD functionality implement 2 resilience control and in a particular scenario one of this control has no effect, then the Database will override the value of “Resilience control” to 2.

In the following E-R diagram for the Domain dependent Library is reported, with highlighted in yellow the parts that can override the information already included in the Ontology.

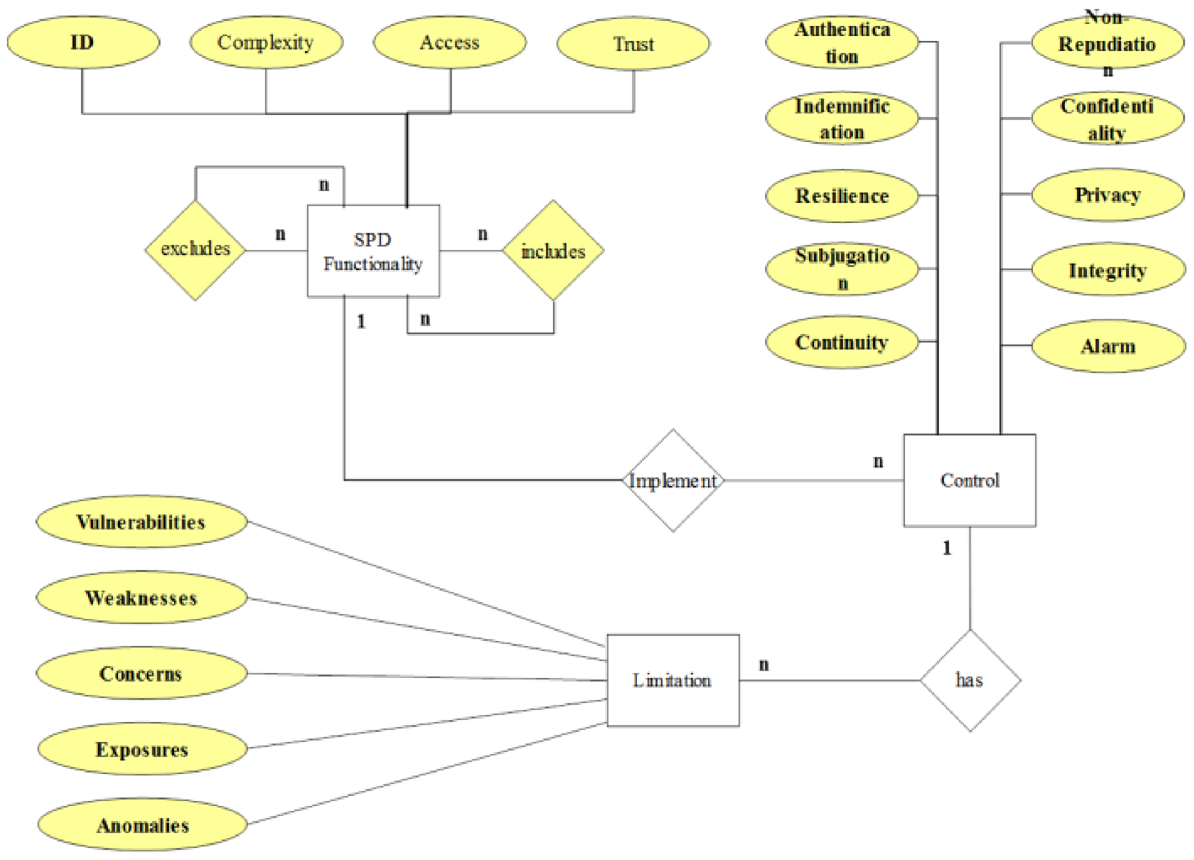


Figure 2-14: SHIELD Domain Dependent Library E-R Diagram

This Library can be also referred to as “context” and is used by the Overlay during the composition process in this way:

Step 1: at first the Ontologies are retrieved by means of discovery services.

Step 2: Ontologies are updated by means of context information

Step 3: the inclusion dependencies are evaluated as well as the exclusions one and the solution is updated accordingly

Step 4: the solution for the composability problem is computed

Elaborated composition algorithms that will leverage this formulation will be analysed in the prosecution of the project.

2.2 Semantic Intrusion Detection

In this section some analysis on a Semantic Intrusion Detection are reported

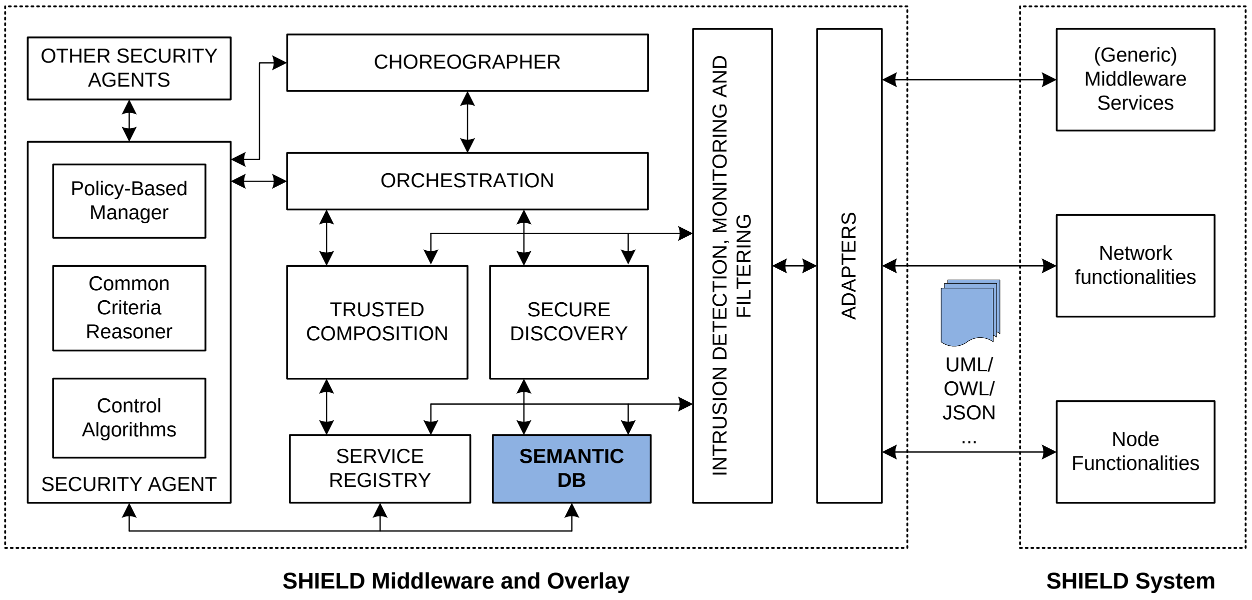


Figure 2-15: Semantic Intrusion Detection

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. [14] 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.

They also cite the following characteristics as an advantage to more traditional methods:

- Grasping the knowledge of a domain: the domain knowledge can be captured by domain ontology.
- Expressing the intrusion detection system much more in terms of the end-user domain: by using the domain ontology, the design of the intrusion system can be expressed in terms of the end-user's domain.
- Generating the intrusion detection system more easily: from the knowledge given in the domain ontology, it is possible to derive a number of properties for an object.
- Making intelligent reasoning: it is not easy to make intelligent reasoning from a scene to the other one. However, it is possible to do that using ontology.

Undercoffer et al. [15], [16] 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.

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. [14]
- The work from Abdoli and Kahani [17]
- The work from Isaza et al [18], and [19]. These authors not only define an ontology for IDSes but they also follow a methodology [20] called METHONTOLOGY to develop the ontology.

Isaza et al. affirm that they use METHONTOLOGY 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 incorporates the description of the attack taxonomy, but also axioms and rules describing the attacks.

The specification phase in this methodology includes asking a set of competence questions (CQ) for the domain to allow building the ontology in an easier, faster way. Questions such as:

- What is the events sequence that describes an attacks type?
- What kind of reaction must be assumed as a result of a possible attack?
- What impact does an attack on the underlying distributed environment?
- What kind of attacks requiring priority reactions (high, medium, low)?

From these questions, not only the entities and their attributes are built into the ontology, but also axioms and rules describing the attack types.

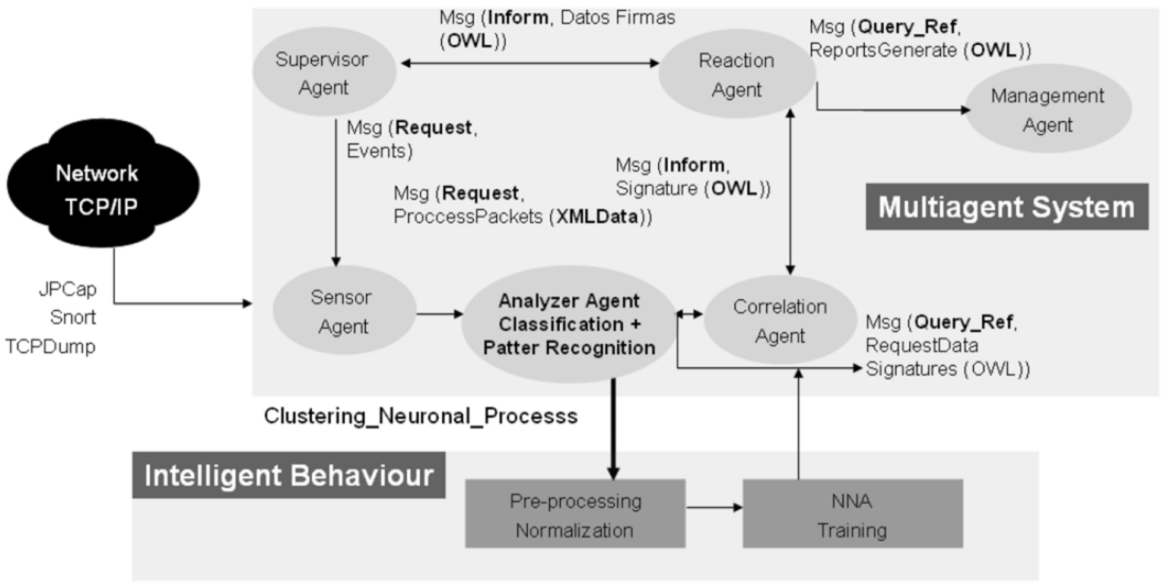

Msg (**Inform**, Datos Firmas (**OWL**))
Supervisor Agent
Reaction Agent
Msg (**Query_Ref**, ReportsGenerate (**OWL**))
Management Agent
Msg (**Request**, Events)
Msg (**Inform**, Signature (**OWL**))
Network TCP/IP
Msg (**Request**, ProccessPackets (**XMLData**))
Multiagent System
JPCap
Snort
TCPDump
Sensor Agent
Analyzer Agent Classification + Patter Recognition
Correlation Agent
Msg (**Query_Ref**, RequestData Signatures (OWL))
Clustering_Neuronal_Processs
Intelligent Behaviour
Pre-processing Normalization
NNA Training

Figure 2-16: OntoIDPSMA MAS architecture

Isaza et al. define a Multi-Agent System (MAS) for IDS and IPS architecture, which they call OntoIDPSMA – Ontological Intrusion Detection and Prevention Multi-agent system. The main roles in the agents that participate in the Multi-Agent System (MAS) model are:

Sensor Agent

- *Intentions*: Capture packets from the network and send them to other agents to be analysed and processed.

<u>*Analyser Agent*</u>

- *Intentions*: Receive data from Sensor Agent, minimize false positives and false negatives, and compare signatures with predefined patterns or behaviours.

<u>*Correlation Agent*</u>

- *Intentions*: Aims to the integration, classification and correlation from events and alarms stored in the Ontology model, using reasoning tools applied in ontological models.

<u>*Reaction Agent*</u>

- *Intentions*: Manages the events to generate alarms and to create a prevention model integrating reaction rules to reconfigure other network devices.

The following figure shows the interaction and the OWL message exchange used by the agents in the OntoIDSPMA architecture, as well as the performative parameter ACL Message and action.

They use Artificial Neuronal Networks with a supervised learning for the intelligent component behaviour. In that component a normalized process for packet captures has been used; the relevant fields are classified and coded to binary. The fields information used are ToS (Type of Service), Length, TTL (Time to Live), Sequence, ACK, Flags, TCP and Data content.

For the ontology rules are defined that allow properties inferences and reasoning process. The IntrusionState, WebAttack, SQLInjection, BufferOverflow, DoS, dDoS, PrivilegedAccess properties among others, describe the anomaly behaviour. They are defined as ontology's attributes, from the captured and processed attack instance using the detection engine, identifying the Type of Intrusion. These values value are imported into the ontology through format conversion sequences from the original data capture formats (IDMEF XML messages, Pcap/tcpdump captures, etc.).

To define the rules Semantic Web Rule Language (SWRL) is used. SWRL is an expressive OWL-based rule language that allows writing rules expressed in OWL concepts and provides some capabilities. A SWRL rule contains an antecedent part to describe the body, and a consequent part, referring the head.

An example for a SQL Injection Rule that describes the intrusion state (p) directed to node (z), with source host (x) is given by:

```
           (NetworkNode(?x) ∧ NetworkNode(?z)
       ∧ IntrusionState(?p) ∧ GeneratedBY(?p, ?z)
      ∧ SQLInjection(?p) ∧ Directed_To(?p, ?z)) →
             SystemSQLInjectionState(?p, ?z)
```

We can also generate a value attribute in the inference process a part of a rule evaluation. For example, if we have a Web Attack Rule like the following one, the target host (x) state is defined as True for Under Web Attack, given the axioms, sentences and conditions that meet the specified claim:

```
           (NetworkNode(?x) ∧ IntrusionState(?y)
        ∧ GeneratedBY(?x,?y) ∧ GeneratedBY(?x,?y)
          ∧ WebAttack(?z) ∧ AttackTypeOf(?y,?z))→
                 UnderWebAttack(?x, ?true)
```

A possible description for the axiom that describes a RootAccess state is denoted as:

```
RootAccess ≡        ∍ (IntrusionState ∩ InputTraffic ∩ NetworkNode)
    ∩ ∍ Generated_by(AnomalousSSH ∪ AnomalousFTP ∪
                    WebAttack ∪ TelnetAccess)
    ∩ UID_Try(UID_0)
```

As a conclusion, the ontologies and reasoning systems developed by different authors so far offer several advantages over more traditional approaches, like an increase in the performance in tasks such as distribution of the knowledge among nodes, intelligence reasoning, knowledge representation, generating inferences, and adaptability, among others. Additionally the correlation model, if present in the architecture provides scalability to the ontology providing a more semantic approach to the model.

However, the CPU usage at each node is greater due to interaction with multiple tools, virtual machines processing and ontological management. This is why we propose not to use this Intrusion Detection Model in the nSHIELD demonstrators, given that the micro/personal nodes are quite constrained in their CPU capabilities.

3 SHIELD middleware core SPD services

3.1 SHIELD secure service discovery and delivery

In this section the SHIELD secure service discovery and Delivery are depicted.

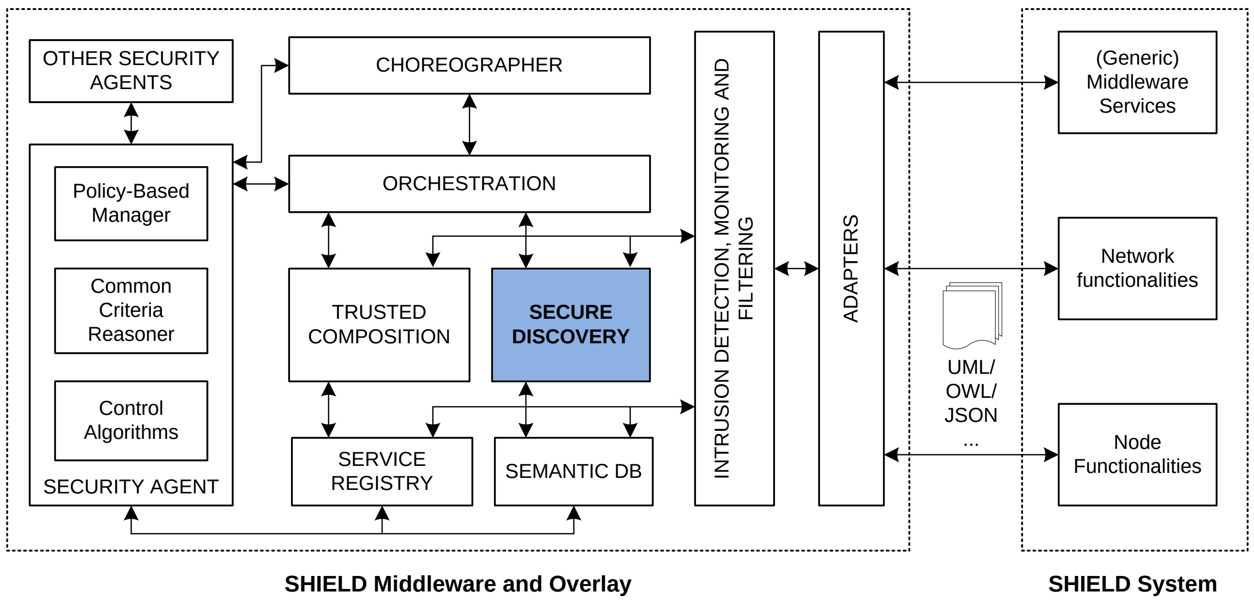


Figure 3-1: SHIELD secure service discovery and Delivery

3.1.1 Service Discovery concept

The **Service Discovery** (SD) is a lookup operation accomplished by a generic entity in order to identify all available services which satisfy specific criteria and requirements. The Service Discovery can be done either manually or automatically by software or by an automatic device.

A **Generic Service** is characterized by a location (physical and logical) and some other specific properties, which allow to identify it by an entity. In general, it is necessary to compose a careful query with the useful parameters to identify an appropriate service. For example:

- the type of service;
- the physical location;
- the capability;
- the delivery method;
- the cost;
- …

and the other values which the user claims from the service. If there exist one or more services the respects this requirements, it or they notify the user with their availability.

Even though the Service Discovery concept is very simple to understand, it is also one of the greatest barriers for the optimal use of the modern technologies. In fact, nowadays, we are surrounded by thousands of programmable devices able to offer several services but, in the same time, often mutually incompatible.

The incompatibility has a direct effect: to take advantage of advanced devices, the user must use different technologies manually, each with the own interfaces, the own services, the own configurations … as well as to compose these services respecting their own requirements and paying attention to the compatibility problems. It is clear that these operations require also the broad expertise of the user.

Internet is the exemplary case study to understand the utility of the Service Discovery: in fact, Internet connects a huge number of customers and services from the whole world and it is clear the importance of the search engines to find the required information. We can see a search engine as a simple Service Discovery which search in the web documents, pictures ore complex e-commerce services.

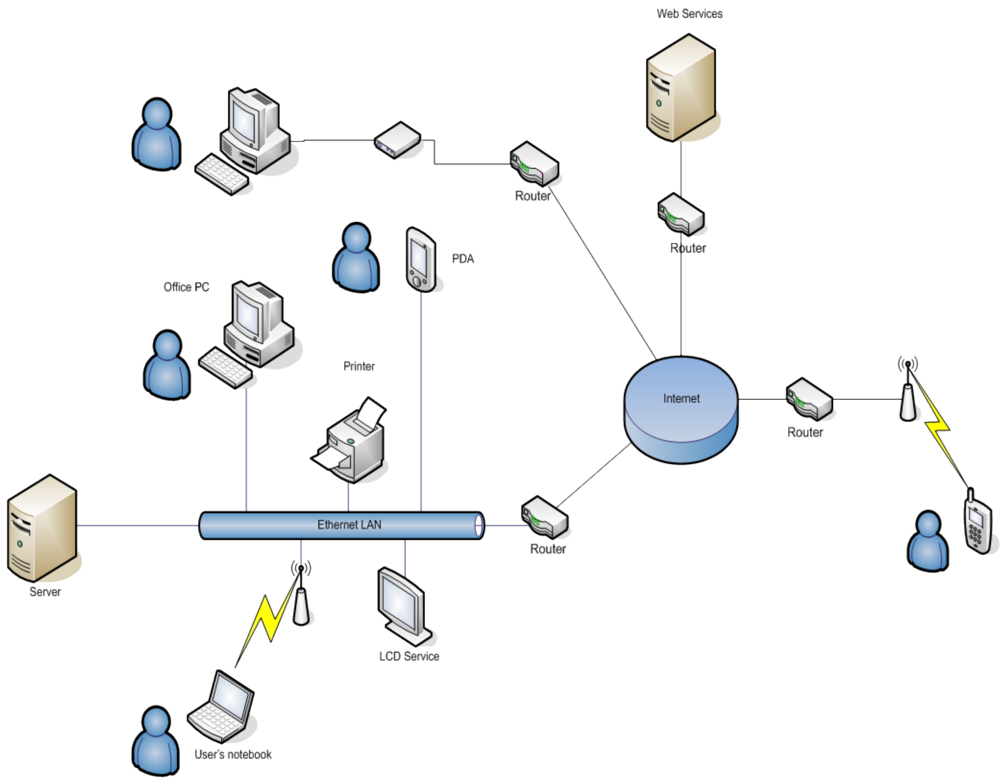


Figure 3-2 – Example of Service Discovery architecture

The concept of **Service** is more general. A feasible definition could be: "*anything which can be used by someone*". A service can be a printer, a projector, generic information (written or audio or video), a call, a video call conference, the access to the appliance control panel or to the alarm system …

Does not exist a unique and absolute definition of Service: it depends from several factors and, in each scenario and in each context, a class of services exists and allows to describe all the relevant characteristic. It is obvious that, the heterogeneity of the information will be one of the most important problems to be addressed.

3.1.2 Overview of Service Discovery protocols

In the last ten-fifteen years, more effort was dedicated to the development of Service Discovery Protocols (SDP). For a deep analysis we suggest to read the papers [21] and [22]. In this section, we illustrate the Service Discovery Protocols used to develop nSHIELD prototype.

3.1.3 Service Location Protocol (SLP)

The Secure Location Protocol (SLP), created by the Service Location Protocol Working Group (SVRLOC) of Internet Engineering Task Force (IETF) and referenced in SLPv2 standard ([23] and [24]), is one of the more used Service Discovery Protocol. In this protocol are considered three basic entities, named "Agent":

- Service Agent (SA);
- Directory Agent (DA);
- User Agent (UA).

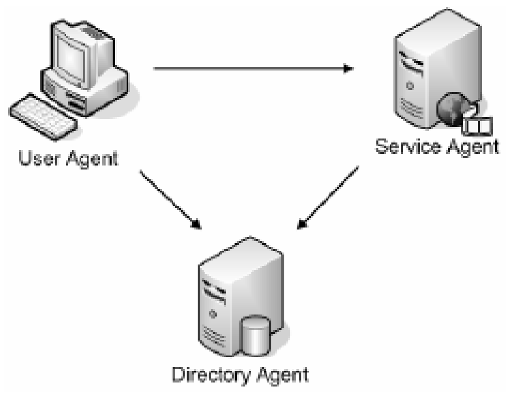


Figure 3-3: SLP logic architecture.

It is a simple, scalable, light and decentralized protocol; furthermore it is independent by HW, SW and the language programs.

A SLP service has some "properties" that describe it. The first is the Service URL: it is a string with a specific form, and it specifies the general category of the service that it describes. For example, a Service URL may be "service:content-adaptation:sip://can1@nshield.eu": it says that the current service is a content-adaptation service, that may be reached, with SIP protocol [25], at the SIP address can1@nshield.eu. The "service:" is a fixed string that says only that following is the URL of a service.

Each service, beyond a Service URL, has a list of Attribute-Values couples. Each Attribute is a property of the service, and it is indicated by a name. This property, typically, has one or more values: so a couple can be "Supported_Resolutions = 640X480,800X600,1024X768". This attribute indicates that that service (that can be a monitor or a projector) has an attribute, named "Supported_Resolutions" (that is auto-explicative) that can assume 3 possibly values: 640X480, 800X600 and 1024X768.

So, supposing to describe a projector as a service on a LAN, it Service URL can be something like "service:video-ouptut-device:projector://p1" (supposing that a protocol exists that uses the projector://p1 as a way to indicate an address of a projector) with the "Supported_Resolution" attribute mentioned above.

After this introduction, we can enter in the SLP architecture's detail. As said previously, SLP uses three kinds of entities: Service Agent (SA), User Agents (UA) and Directory Agents (DA). The SA is the service supplier: it has to register its services on the DA. The DA is the "core" of the architecture, because it registers all the services that are offered by a network. Finally, the UA is the client that interrogates the DA to find a specific service of a SA. See Figure 3-3.

On the DA, for each service, beyond Service URL and its attributes (with their values), there is stored a Service Scope (that is "visibility area"), a Service Type (that is indicated the Service URL) and a Service Life Time (that is a time that, after expired) causes the service to be unregistered. This additional information are used to simplify the discovery process and to make failure-safe the entire architecture:

indeed, if a service crashes it can’t renew its registration on the DA, the Life Time expires and it is removed; so no one DA can find that service further!

The service discovery is done as follows. The UA sends a message to find a service. This message can be either in unicast mode to a specific DA, or in multicast: in this last case, each DA or SA that is on the network receives that message. If there are only SAs, the ones that satisfies to the UA requests, respond to it; otherwise, each SA that respond to the UA requests and each DA where is registered a service that satisfies to the UA requests, respond to the UA. So, SLP can work both in a centralized and in a decentralized way. The UA message contains a visibility area, a service type and a list of capabilities that the service has to comply. All messages exchanged by entities are extensively defined in SLP specifics: these messages implement a robust asynchronous communication protocol.

The branch where SLP excels (on other protocols above and below reported) is in its capability of perform “complex” queries on the attributes’ values: it can use Boolean operators (AND, OR, NOT), comparators ($<$, $\leq$, $>$, $\geq$, $=$) and functions of string matching.

The motives because SLP was the protocol chosen for this thesis are summarized below:

- it has the best query language and capability
- it is highly scalable: with service Scopes and the possibility to work either in a centralized and in a decentralized way, it can perfectly work either in small LANs and in the Internet with the same simplicity
- it supports the possibility of using more DAs, hierarchically interconnected: with SLP it is possible to create a service discovery architecture like the DNS one
- it uses a moderate number of multicast messages and has a very low traffic overhead

Summing, we can consider two type of messages exchange in SLP: one is for a registration of a new service and the other for service requests.

New Service Registration

1. the SA sends a service registration request (*SrvReg*) to the DA when it want share an own service;
2. the DA registers the service and replay to the SA with a service acknowledgement message *(SrvAck)*;

Service Request

1. the UA sends a service request (*SrvRqst*) asking the type and the parameters of the desired service;
2. the DA checks for the UA request and answer with a service replay message (*SrvRply*) including the address of the services required.

```
+-------+ -Unicast SrvRqst-> +-----------+ <-Unicast SrvReg- +-------+
| User  |                    | Directory |                   |Service|
| Agent |                    |   Agent   |                   | Agent |
+-------+ <-Unicast SrvRply- +-----------+ -Unicast SrvAck-> +-------+
```

Figure 3-4: SLP flow for service registration (upper line) and require (lower line).

3.1.4 SLP Message

In order to introduce the security policy of SLP protocol, a little summary of the main SLP messages is required. The messages exchanged in SLP protocol have all the same header.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Version    |  Function-ID  |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Length, contd.|O|F|R|       reserved          |Next Ext Offset|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Extension Offset, contd.|              XID              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Language Tag Length      |         Language Tag          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-5: SLP header

We report now the most important fields of the SLP header

3.1.4.1 Function-ID

Table 3-1: ID value of the SLP header

Type Message	Acronyms	Function-ID
Service Request	SrvRqst	1
Service Reply	SrvRply	2
Service Registration	SrvReg	3
Service Deregister	SrvDeReg	4
Service Acknowledge	SrvAck	5
Attribute Request	AttrReq	6
Attribute Replay	AttrRply	7
DA Advertisement	DAAdvert	8
Service Type Request	SrvTypeRqst	9
Service Type Reply	SrvTypeRply	10
SA Advertisement	SAAdvert	11

This field report the unique code of the type of the message. The allowed IDs are shown in Table 3-1.

3.1.4.2 Length

This field report the length of the whole message included the header (it is a integer value).

3.1.4.3 Flags

This binary field allows to set the option of the transmission mode:

- Flag O (Overflow): is “1” if the message length exceeds the length of the datagram;
- Flag F (Fresh): is “1” for each new registration message;
- Flag R (multicast Request): is enabled if the request must address in multicast mode.

3.1.4.4 XID

This is a unique number for a request which allows to identify the thread. So, the following replay messages keep the same XID value.

In the follow, we report the more important message exchanged in SLP protocol.

3.1.4.5 Service Request

The ‘Service Request’ message is sent from the UA to the DAs to retrieve already registered services. Figure 3-6 shows the structure and, follow that, the main fields of the messages are described.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Service Location header (function = SrvRqst = 1)        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      length of <PRList>       |        <PRList> String        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   length of <service-type>    |    <service-type> String      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    length of <scope-list>     |     <scope-list> String       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  length of predicate string   |  Service Request <predicate>  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  length of <SLP SPI> string   |       <SLP SPI> String        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-6: Service Request message.

- **<PRList>** (Previous Responder List): empty in case of unicast requests; it contains the IP addresses of the DAs found till that moment in case of multicast requests.
- **<service type>**: the type of the service sought.
- **<predicate>**: optional. It is an additional filter on the attributes of the services to be found.
- **<SPI>** (Security Parameter Index): the security parameters with which the UA was configured; if omitted, the reply message has not to contain any Authentication Block. If the SPI are not supported by the DA and AUTHENTICATION UNKNOWN errors returned.

3.1.4.6 Service Reply

An UA receives this message by a DA as a response to its request for services: such a reply contains, further than the usual header, an *Error Code* indicating there were errors if its value is not zero.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Service Location header (function = SrvRply = 2)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Error Code             |        URL Entry count        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       <URL Entry 1>          ...       <URL Entry N>          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-7 – Service Reply message

The < *URL Entry count* > field indicates the number of services found; subsequently there is a list of URL Entry, each one having the form shown in Figure 3-8.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |          Lifetime             |   URL Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|URL len, contd.|            URL (variable length)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|# of URL auths |            Auth. blocks (if any)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-8: URL entries

As previously said, each service has also a 'Lifetime', i.e. the time on which the service is active from its registration.

3.1.4.7 Service Registration

This message is sent by the Service Provider (SA) in order to register a service on a DA, specifying all the characteristics of the service, such as its type, its attributes and relative values, the lifetime, etc.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Service Location header (function = SrvReg = 3)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          <URL-Entry>                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| length of service type string |        <service-type>         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     length of <scope-list>    |         <scope-list>          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  length of attr-list string   |          <attr-list>          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|# of AttrAuths |(if present) Attribute Authentication Blocks...|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-9: Service Registration message

The last field is the Authentication Block (optional), used for secure the SLP messages: it contains the digital signature of the SA and all the parameters to be used to verify the message after its reception.

3.1.4.8 Service Acknowledgment

The *Service Ack* is sent by the DA to the SA as a response to a Service Registration, in order to inform the SA of the right registration of the service. The Error Code will be different by zero in case of errors.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          Service Location header (function = SrvAck = 5)      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          Error Code           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-10: Service ACK

3.1.4.9 Directory Agent Advertisement

This message is sent by the DA, containing all the information about it.

The field < *Error code* > is the usual; the field < *DA Stateless Boot Timestamp* > indicates the status of the DA. It is zero to indicate the DA is not active from that moment. The message could also contain a list of the attributes of the DA. The <*scope-list*> is the list of the scopes supported by the DA.

Among the attributes concerning the DA, particularly important is the < *min-refresh-interval* >, indicating the minimum time among two information refreshes from a SA for a particular service. The field URL contains the string "*service:directory-agent://<addr>*", where *<addr>* is the address of the DA. Finally, the < *SPI List* > contains the list of security parameters the DA is able to verify.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |        Service Location header (function = DAAdvert = 8)      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          Error Code           |  DA Stateless Boot Timestamp  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |DA Stateless Boot Time,, contd.|         Length of URL         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                              URL                              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Length of <scope-list>    |         <scope-list>          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Length of <attr-list>     |          <attr-list>          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |    Length of <SLP SPI List>   |     <SLP SPI List> String     |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 | # Auth Blocks |         Authentication block (if any)         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-11: DA Advertisement message.

3.1.4.10 Error Code

More messages, also among the previously, have a <Error Code> field. This field is composited by two byte and indicates if there is an error in message exchange. If this field has a value different from zero, the rest of the message could be trunked. Only the error occurred in unicast requests are returned, instead the messages are simply discarded in multicast requests. In --- are shown the main error.

Table 3-2: Main Error code in SLP message.

Type of Error	Code	Description
PARSE ERROR	02	Syntactical Error.
INVALID REGISTRATION	03	Registration problem (ex. <*lifetime*>=0 or it lacks a field).
SCOPE NOT SUPPORTED	04	The <*scope*> is not supported by DA or SA.
AUTHENTICATION UNKNWON	05	The security parameters are not supported.
AUTHENTICATION ABSENT	06	It lacks an <*Authentication Block*>.
AUTHENTICATION FAILED	07	There is error signature verification.

3.1.5 Security in SLP

In this section, we discuss the issue of the SLP security and, in particular, of various specifications that the protocol introduces to ensure a specific level of security. Also, we discuss the main method including in this protocol and, finally, the changes introduced in order to improve the security level.

The security issue is addressed in order to allow the customers to use services which are not included in the own network. The sharing of services of different network is an important problem to treat both for costumer and for network owner security. SLP has the possibility to authenticate some messages coming in the network. This process is based using a public authentication key and it certificates the trusted identity of the sender agent. An agent sends a message including a digital signature starting from the original message. The only message where this is possible are the message which include the <*Authentication Block*>, see the Section 3.1.4.

The security process requires two key: the first is private and it is used to compute the digital signature and other is public and it is necessary to verify it. The relation between a DA and a SA is established, according with SLP standard, in configuration phase when there is the exchange of keys. <u>The main problem of this security approach is that the trusted relation can occur only in configuration mode</u>.

Moreover, there are other two problems in this security scenario: the first regards the DA, because it is not possible guarantee that a generic UA does not access to the high protection services; the second concerns the possibility of an UA to ascertain that a DA is trusted. The first problem is not dealt whit a traditional SLP standard, instead the second can be resolved guaranteeing that the public key is exchanged in security mode with the DA.

The following paragraphs show how this latter happens.

3.1.5.1 Authentication Block

It is important to introduce the <*Authentication Block*>. Its structure is shown in Figure 3-12.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Block Structure Descriptor   |  Authentication Block Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     SLP SPI String Length     |         SLP SPI String        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Digital Signature                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-12: Authentication Block structure.

The <*Authentication Block*> drives some SLP messages in order to guarantee the originality of the message and that the source is a trusted identity. Beyond the length fields, there are four important fields:

- BSD (Block Structure Description) which identifies the algorithm used to compute the digital signature. The default choice is the DSA algorithm [26] using SHA-1 (Secure Hash Algorithm) hash function [27], but it is not the only.
- Timestamp: it indicates the expiration time of the signature.
- SPI (Security Parameter Index): it contains the parameter useful to understand and to verify the digital signature, such as the key length as well as the BSD parameters or the public key.
- Digital Signature: in this field is written the signature.

3.1.5.2 Signature generation process

The generation process of the signature is divided in two phases: in the former it runs the hash of the message and in the latter it applies the algorithm.

The hash functions are particular algorithms which cover a primary role in the signature process. The main algorithms take input messages with fixed length, but this implies several problems for long messages. In fact, one possible solution could be to divide in small blocks but it entails other problems as:

- the space, the signature length is proportional to the blocks number;
- the time, the checking of the signature requires several and complex operations;
- the security, sniff and replicate the messages is simpler for intruders.

The use of a hash function provides an efficient solution to all these problems. In particular, it accepts input messages of any size and it return a fixed length messages which acts as fingerprint. This approach guarantees that if the original message will change, also the hash function will be different.

The fingerprint of the message is given in input to an algorithm which creates the digital signature, using and combining also the private key.

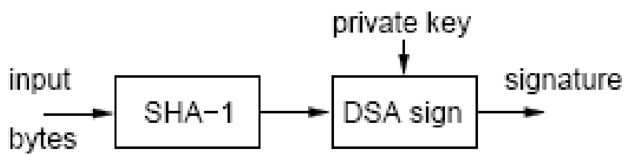


Figure 3-13: Signature generation process (according with DSA + SHA-1 standards)

3.1.5.3 Signature verification

Once a message with Authentication Block is received, the receiver must extract the BSD and the SPIs in order to identify the algorithm used to sign the message and relatives parameters as well as the public key.

Firstly, the receiver applies the hash function (the same used by the sender) to the message, then it decodes the message using the public key and, finally, the hashed and the decoded message will be compared so that it is possible to verify the correctly of the signature.

If there is one or more problem, the message will discard and the receiver will sent and error message to the sender using the <*error code*> already illustrated in Paragraph 3.1.4.7.

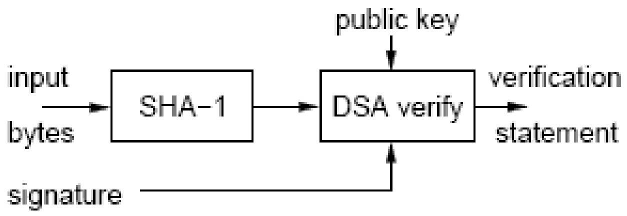


Figure 3-14: Signature verification (according with DSA + SHA-1 standards)

3.1.5.4 Security scenarios

In the previous paragraphs, we have described how it is possible to assure the authenticity of the messages exchanged in SLP protocol. The signature process allows to verify if the message are authentic and, also, if the message is intact. All this thanks to hash function, because if a message is changed, for any reason, also the hash signature must be different and the verifier will deny the message received.

Unfortunately, however, the SLP messages are not secure. In fact, the messages are sent in clearly mode and, so, anyone could eavesdrop and read. To introduce this protection, it is necessary combine SLP message with other high-level security protocol which allows, for example, cryptography.

Hereinafter, we illustrate some important scenario where security attacks typically happen and the countermeasures implemented in nSHIELD project.

The main threat for SLP is due to the information which are eavesdropped and stored in the network in order to be sent in other moment (*Replay Attacks*).

3.1.5.4.1 Scenario 1

We consider an exchange of message between a Directory Agent and a Service Agent as shown in Figure 3-15.

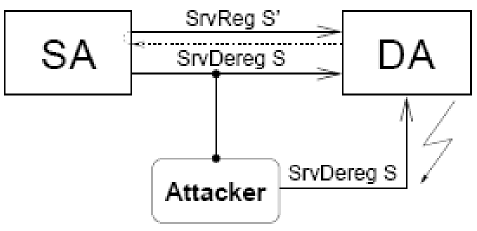


Figure 3-15: Example of *Replay Attack* (Scenario 1)

The SA sends a message to DA in order to delete a generic service S from its cache, using *SrvDereg* structured request. An Attacker intercepts and stored it. When the SA will send a new require for service registration (*SrvReg*), the Attacker could re-send the intercepted message, pretending to be the SA.

3.1.5.4.2 Scenario 2

Another scenario is shown in Figure 3-16. At time t_1, the SA registers a service S which will end at time t_4. An Attacker intercepts and stores this message in its memory at time t_2, the SA wants to change some service parameter, including the lifetime in such a way to delay it at time t_5. At time t_3, if the Attack sends the message stored at time t_1, the second message will be deleted and the service will reset at time t_4.

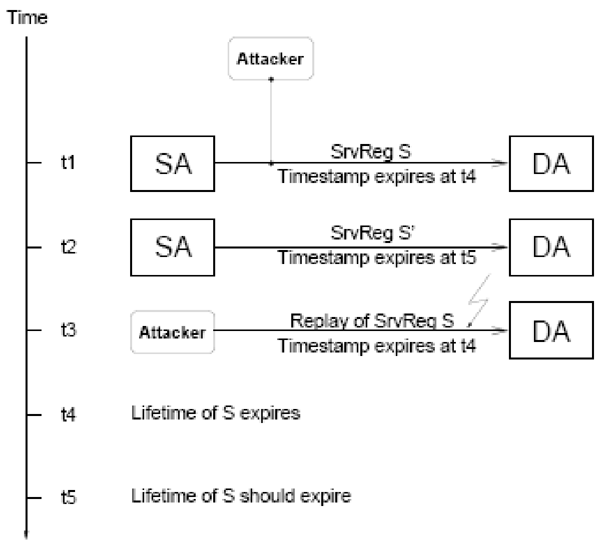


Figure 3-16: Example of *Replay Attack* (Scenario 2)

3.1.5.4.3 Scenario 3

Following the previous scenario, this has some different that particularize other aspects. As previous, at time t_1 a SA register a service S (such us a printer service) but, in this case, it expires at time t_{10}. At time t_5, the network administrator decides to remove this service from the DA.

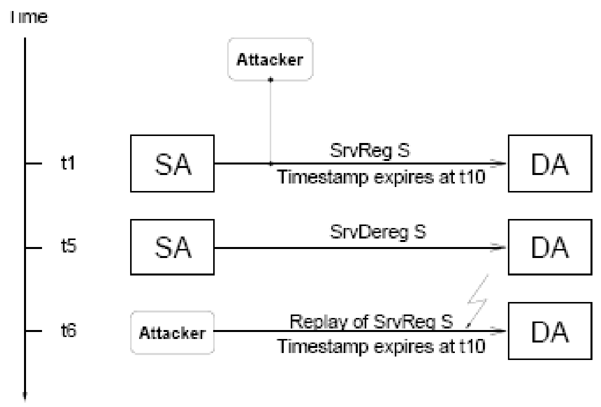


Figure 3-17: Example of *Replay Attack* (Scenario 2*)*

So, the Attacker has the interval time from t_5 to t_{10} to pretend itself as a SA (for example using the spoofing IP). During this interval, the Attacker will receive all documents that all users think to send to the printer service.

These scenarios and threats are been dealt introducing small improvements at the traditional SLP. In particular, the approach developed follows the criterions:

1. the DA take notes of all timestamps included in the Authentication Block of the messages received;
2. to transmit a new message, the SA must use ever a timestamp t'' higher than that of the previous (t').

So, the DA compares the timestamp of the last message received and if it is less than that of the previous, the DA discards this latter. The DA will accept and store only the messages which have t''>t' (Figure 3-18). It is simply to verify that this small shrewdness solves almost all the *Replay Attack* seen in this paragraph.

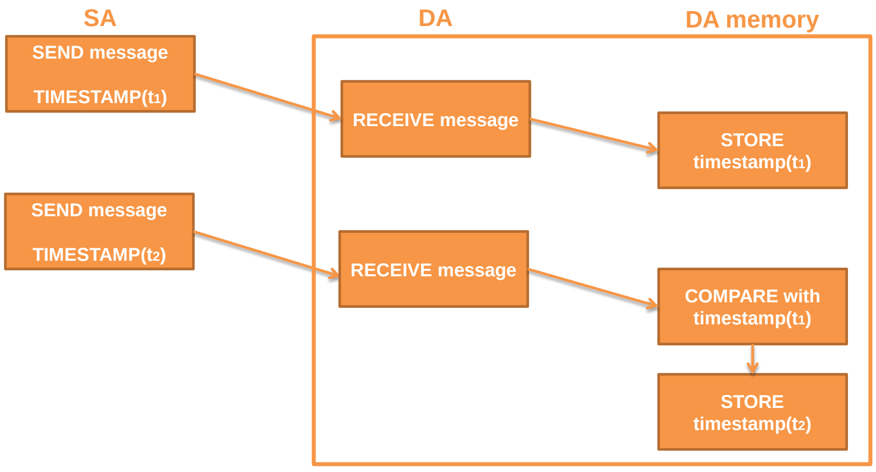


Figure 3-18: Anti-*ReplayAttack* flowchart 1

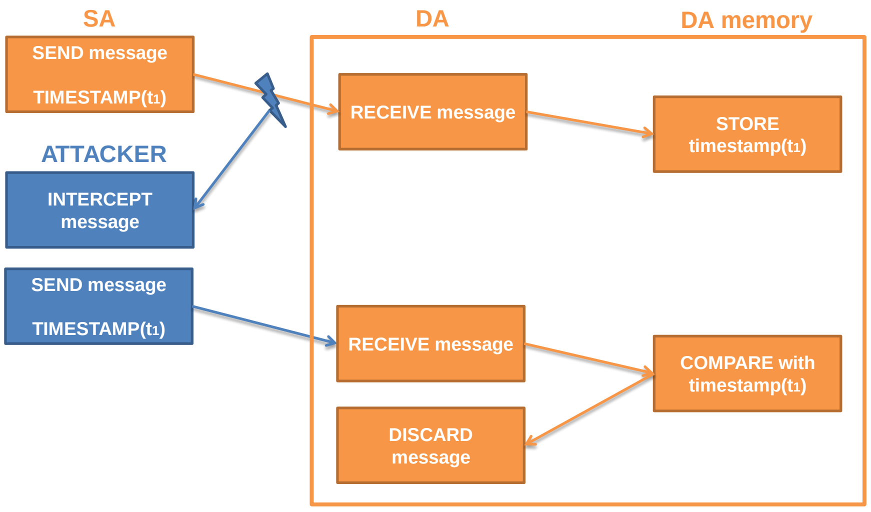


Figure 3-19: Anti-*ReplayAttack* flowchart 2

3.1.6 Secure Architecture for Service Discovery

The service discovery process of a device is a service offered by the network. One of the main aspect to take attention is the security due to the heterogeneous environment and, primarily, because the service discovery network is very useful when all devises can access at the network. To deal this problem, in nSHIELD project, we have exploited the SLP protocol implementing relevant improvements. As previously illustrated, SLP has few shrewdness to guarantee secure discovery. It was necessary to introduce other specifications in order to allow nSHIELD to discovery service securely.

Summarizing, the SLP signature process is as shown in Figure 3-20 and, for more detailed specification we suggest to read Section 3.1.3.

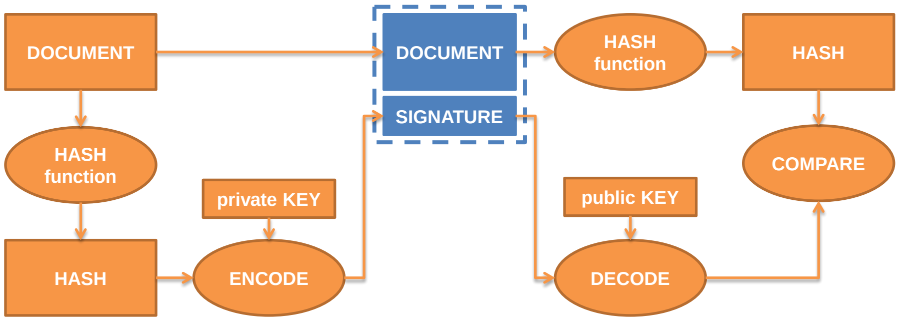


Figure 3-20 – SLP flowchart

The SLP approach allows to guarantee some security specification such as the authentication and the data integrity. Whit some improvement, in paragraph 3.1.5.4, we show how to prevent *Replay Attack*, but it still remains the confidentiality problem. In fact, all devices connected in a network could read the SLP messages exchanged among all SAs and all DAs.

The only parameters on which the SLP bases own protection are in Authentication Block and, in particular, the field named **SPI (Security Parameter Index)**. The main role of this field is to give information to the received entity for doing the signatures verify. For further details, you can see paragraph 3.1.4.

To guarantee the confidentiality communications for service discovery we have introduced the **PKI (Public Key Infrastructure)**. Generally, it is a set of devices, such us hardware, software … but not only, it could be also people, policies, or procedures … anything skilful to create, manage, distribute, use, store, and revoke digital certificates. For our use, we consider it is an trusted certification authority able to issue, on demand, a certificate which guarantees the trusted relation between the owner and a public key. Only than the certificate will be published and shared with others. The whole process it can be described in three steps:

1. initialization of the PKI (not if it already exist);
2. certification (request and issue of the certificate);
3. authentication (signature and verify)

Our approach extends the traditional SLP standard, deleting the pre-configuration phase where there were the recognition and federation phases using just the PKI. In this manner, we high the level of security because to have a public key it is necessary to register into trusted PKI. Not only. This allows to improve transparently the Public Key Infrastructure as soon as possible to have a better system. The follow paragraph deals the iteration between nSHIELD and the PKI.

3.1.6.1 Iteration between PKI

This paragraph is dedicated to illustrate how the nSHIELD prototype communicates with the PKI and the reasons to use a PKI. The SLP standard implementation needs a setting phase where the entities involved change the own key, code/encode parameters and other preferences. Using PKI, this pre-configuration step is useless, or better, it is declined to it, entirely.

As before just mentioned, the process can be summered in three step:

1. initialization of the PKI (not if it already exist);
2. certification (request and issue of the certificate);
3. authentication (signature and verify).

3.1.6.1.1 Initialization of the PKI

The means which a PKI uses to associate a generic entity with a public key is the digital certificate. The entity which emits it is the **Certification Agency (CA)**. Nowadays, the CA are more diffuse and they are used to guarantee a defined level of internet security, particularly. The main applications are to avoid frauds in the on-line bank or other type of electronic commerce, and to allow a secure communication among all web-sites which require reliable services. Typically, the process requires that a CA require an own certificate to other CAs, but it is not important for our case. A PKI can be created in local o global network, it is indifferent for the functionalities and depends only by the own policies.

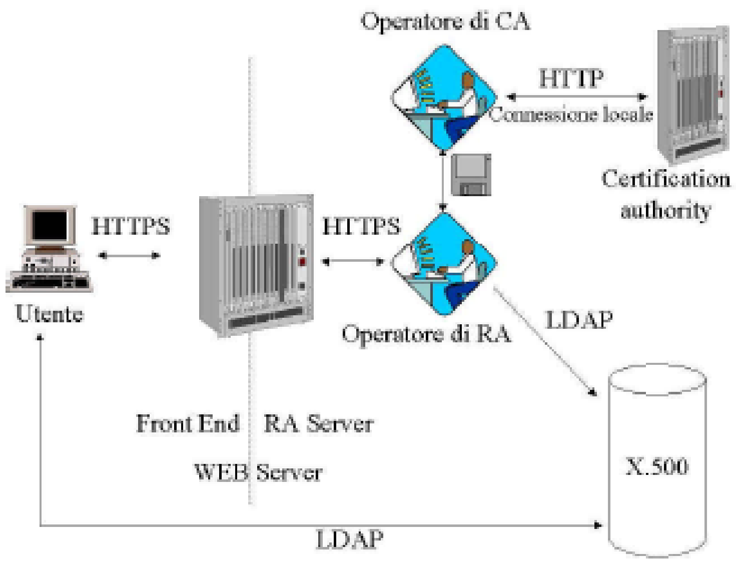


Figure 3-21: SLP simple scheme

In nSHIELD project, we have created a local PKI, but it is also thought for public one. In particular we choose good open source software, named *OpenCA*. It is management software for CA and it is born to offer a toolkit for PKI implementation. It involves four different main entities:

1. The **End Entity (EE)**: the EEs are the owner of the certificates: people, server, router, OID… In our case, they are the Agents of the SLP. The EEs access to the **Personal Security Environment (PSE)** to get the information useful for SLP communication.
2. The **Certification Authority (CA)** that we have described before.

3. The **Registration Authority (RA)**: the RAs work in parallel with the CA in order to recognize the PKI certificate owner and to create a interface for the final user. As well as, when the CAs are servers, it is important decouple the CA with a simple entity that emit only the basic services (i.e. certification request or revoke, or renewal …)
4. The **repository**, which is the public database where are kept the certificates and the polity to access them.

Onetime the main actors are clear; we can entry in the initialization phase. It is composited by four steps:

1. Generation of public and private keys of the CA. This phase requires a password to protect the key just generated.
2. Initialization of database, in particular, the creation of directory, files and tables needed to take note of the certification states.
3. Creation of a own certification which guarantee the correspondence between a CA and the relatives public key.
4. Certification signature using the two keys.

The following phases are similar: both the certification and the registration certificates will create for CA and RA, using similar steps and the password decided previously.

3.1.6.1.2 Certification

The certification is the main peculiarity of PKI: in fact, through this process, an entity can obtain the emission of a certificate which attests the authenticity of the user-public key correspondence. It is useful in two cases:

- Authentication: when it wants to sign and it needs a public key of the signatory to verify it.
- Cryptography: when an entity wants to crypt the messages in order to make visible the content only at who has the public key.

Using *OpenCA*, the procedure to obtain the certificate is as follow:

1. The costumer sets his personal PIN on the CA using the own browser. After that, *OpenCA* generates a couple of keys (public and private). The former is sent to DA, the latter is stored in a Smart Card, or a Token on another device.
2. Periodically, the RA controls if there are new connection requires. If there are, the RA check if each is compliant with the policies defined and, if is necessary, asks other information.
3. The requirements approved by the RA are sent to CA.
4. The CA emits the certificates based on what is written on the requirements.
5. The new certificates are emitted and sent to RA.
6. The RA imports the certificates on own server and updates periodically the **Certificate Revocation List (CRL)** and the **Certificate Suspension List (CSL)**.
7. When a certificate is published, the RA sends a mail to the claimant.
8. The claimant can now download its certificate and works.

This procedure can be done by the three entities which take part at Service Discovery process, that are the Directory Agents, the Service Agents and the User Agents.

The Figure 3-22 shows an example of certification require.

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=IT, ST=Italia, L=Roma, O=SP, CN=Service Provider
        Subject Public Key Info:Service Agent
            Public Key Algorithm: sha1withrsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:ce:0d:cd:08:86:fd:b5:cb:14:56:51:04:73:38:
                    15:77:39:2d:3b:10:17:06:7c:64:0d:69:14:67:cd:
               ...
                    67:f7:ef:b1:71:af:24:77:64:66:64:0f:85:a6:64:
                    16:c2:69:26:59:0a:d9:4b:8d
                Exponent: 65537 (0x10001)
        Signature Algorithm: md5WithRSAEncryption
        8f:25:9f:68:3a:67:4c:6d:e6:eb:52:4a:ca:73:74:47:85:14:
        ca:d6:6c:6d:24:3b:6c:37:59:ec:f8:fb:0b:a9:74:d6:1c:0f:
    ...
        02:60:16:fd:2e:9b:09:af:11:03:82:74:16:ae:57:a7:90:f5:
        e1:a5
```

Figure 3-22: Example of certification require.

3.1.6.1.3 Authentication in the case of Service Registration

The authentication process includes the computation of the digital sign of the message to send. One obtained the certificate, an Entity can sign whatever message because the receiver can decode and reed these using the known key. The SLP is thought for the signature of the message, but there is not any procedure to do it.

Resuming the *Service Registration* message, for example and we consider its structure (Figure 3-23).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Service Location header (function = SrvReg = 3)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          <URL-Entry>                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| length of service type string |        <service-type>         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     length of <scope-list>    |         <scope-list>          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  length of attr-list string   |          <attr-list>          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|# of AttrAuths |(if present) Attribute Authentication Blocks...|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-23: Service Registration message.

The last field allows the possibility to include the digital firm, through the *Authentication Block*, but there is not a standardise procedure. Moreover, the exchange of trusted messages it is possible only if there is a pre-configuration phase, in when the configuration parameters for encode-decode are defined.

The use of Certification Agency avoids this phase. In nSHIELD project, we have chosen to imply the asymmetric algorithm RSA combined the hash function SHA-1.

3.1.6.1.3.1 Hash function

The first step, the message is shaped by an hash algorithm. In particular, a variable size message the SHA-1 function generates a 160 bit string. This message is called *message digest*. It is “secure” because it is computational impossible to find:

- a message which matches at another;
- two different message which produce the same *message digest*.

3.1.6.1.3.2 RSA

The RSA algorithm is used both for cryptography both for the digital signature. The first is based on the existence of two keys and, in particular, it is used the private key for the encode step and the public key to decode the message. If the keys are well-chosen, the two keys are completely independent, in such a way to not allow di find one known the other. The RSA algorithm born on a complex factorization of first number:

1. it is chosen two random numbers, **p** and **q**, mutual independent and fairly large (it is necessary for a good security);
2. it is computed the product **n**=p*q, called *form*;
3. it is chosen a number **e**, called *public exponent,* such that $1<e<\varphi(n)$ and gcd $(\varphi(n), e) = 1$ (gcd is the greatest common divisor);
4. it is compute the *private exponent* **d**= $d = e^{-1} \bmod \varphi(n)$.

```
RSA key, 1024 bits

Modulus n:
152011386056965022080288058704257365115956774034022029605776966206407
054327295788466675009167178031475794360216606168959896588866701550378
146017637962628953861186173877305648320545518682093226197261251037625
546524406468098709550731727497940820614817088944861394933268567783421
008857573534835951194583685267789

public exponent e:  65537

private exponent d:
021078968674140440059575647386791846916192243545740236401653051915475
433052507011288274906321240108889241089072651596800672541478849451568
845688459708386212210709905812638625800656911834802383661308508626925
475135673036881174847930573429459042028595249364093051137729073305920
52349212881578574528929081976193

prime p:
131753684968499383496626954247729538474500742645454820530059455706250
721430051791150993655364964852686459433807614607983616378078069716898
69922710444733041

prime q:
115375434162094969210759645547902500874428137737124360407358372006445
938476683366004452503683391242850597423788817599375601357853450325474
02454408671785629
```

Figure 3-24: 1024bit RSA key example

So, the private key consists on {**d**, **n**}, although the public key is {**e**, **n**}. A typical RSA key is a 1024 bit, such an example in Figure 3-24.

3.1.6.1.4 Signature

After the creation of the hash, the message H(M), where M is the original message in clear, the Service Agent is ready to compute the signature, using the key obtained by RSA algorithm. In particular, it uses the follow equation:

$$F = H(M)^{d} \bmod n$$

Done that, the SA sends a message to Service Registration with the *Authentication Block*, which contains the fields as shown in Figure 3-25.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     BSD = 3 (RSAwithSHA1)     |  Authentication Block Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     SLP SPI String Length     |         SLP SPI String        \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Digital Signature  F                      \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-25: Modified Authentication Block structure

Comparing it with the original (previously shown in Figure 3-12), the changes made on it are in particular on SPI. In fact, to allow the interaction between the PKI and the DA, it was necessary to add the **serial number** between the parameters of the SA certificate.

3.1.6.1.5 Verification

We continue to take in exam the Service Registration message sent by SA. To allow that the DA can read the original message, it is necessary to decode the message received.

The first step of DA is to read the BSD field of *Authentication Block*, in order to understand which algorithm was used to encode and to know the public key. These parameters are sufficient to make a first verification of integrity and the correctness of the key.

Then, it applies the hash algorithm: considering the received message M', H(M') (the *message digest* of M') is obtained thanks to the SHA-1, as in sender procedure.

After that, it is the time to decode the message using the RSA inverted formula:

$$C = F^{e} \bmod n$$

The DA will discard the unsigned message and will process the signed ones. The DA cannot know the reason of sign error.

The second verification step is to verify if the public key and the SA which sent the message are regular. For this, we use the PKI, in fact, the Certification Authority keeps the certificate through the storing of serial number. The same parameter is contained in *Authentication Block*.

1. extract the serial number of the SA certification from the Authentication Block of the message received;
2. connect to CA e require the certificate with this serial;
3. download the certificate, if it is possible;
4. verify the CA signature and the relative public key;
5. download the Certificate Revocation List and the Certificate Suspension List to check if there is any problem.

If it is all ok, the DA informs the SA the successful with a *Service Ack* message.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Service Location header (function = SrvAck = 5)      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Error Code           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-26: Example of *Service Ask* message.

3.1.6.2 Extension of SLP

We propose an extension of SLP in nSHIELD project. In particular, we add two new messages which allow an high level of security in service require phase. The first message (*Secure Service Request*) is for the secure request for a particular service and the second (*Secure Service Reply*) is just the replay at previous request. These extensions allow at the Das to be secure of the authenticity of the UAs which send any type of request.

3.1.6.2.1 Secure Service Request

This first message is based on *Service Request* of traditional SLP, already discussed in paragraph 3.1.4.2. The *Secure Service Request* moreover contains the user sign an, so, it is sent with the *Authentication Block* and relate parameters. The computation and the verification is as previously described in case of *Service Registration*.

This new request type introduces a distinction among services which require authentication or not. This choice is given by the Service Agent: in fact, we add a new parameter among the attributes to understand it. This parameter is "*authentication".* So, if the DA reads that a specific service it distinguee normal and protected services. In the first case the UA will use the traditional *Service Request* otherwise it will use the *Secure Service Request*.

3.1.6.2.2 Secure Service Reply

To increase the secure level of communications and to prevent some types of attacks, we create a new format of *Secure Reply*, too. If an UA send a request using *Secure Service Request* message, the DA will answer with *Secure Service Reply*.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Service Location header (function = SrvRply = 2)      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Error Code             |        URL Entry count        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       ###############################################         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3-27: Secure Service Reply.

3.2 SHIELD trusted service composition

In this section the SHIELD Trusted Composition concept is outlined

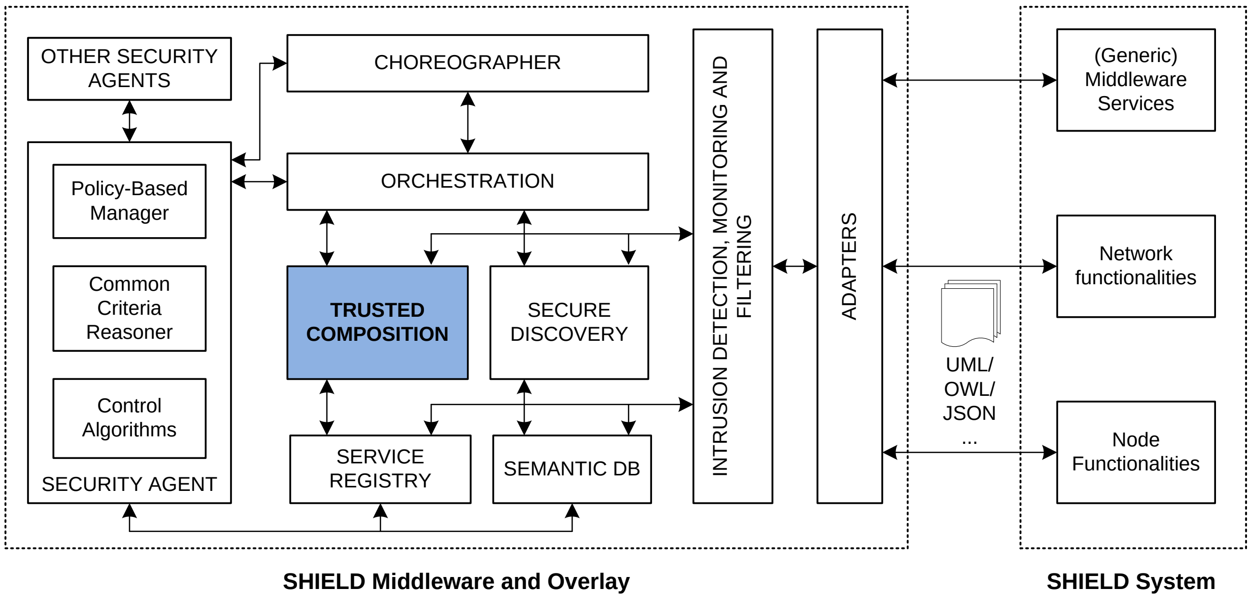


Figure 3-28: SHIELD trusted service composition

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

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

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

In the current deliverable, a preliminary version of the intrusion detection service is provided as a proof-of concept service to demonstrate DDoS protection subsystem that will be used.

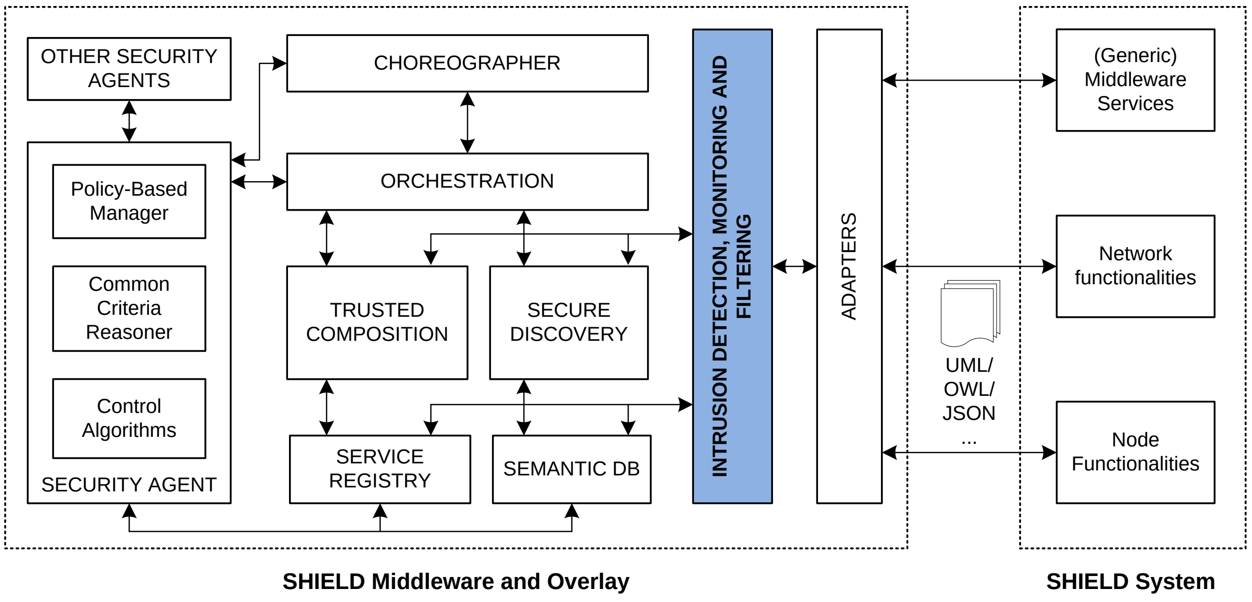


Figure 3-29: SHIELD monitoring, filtering and intrusion detection service

3.3.1 DoS Protection Subsystem

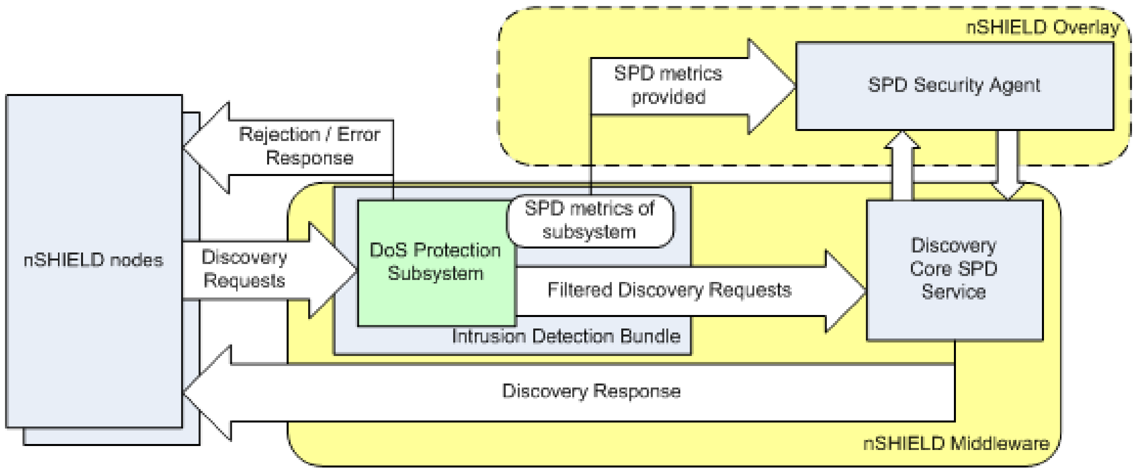


Figure 3-30: The logic block structure of the Intrusion Detection Bundle

The DoS Protection Subsystem provides the core functionality (queuing, filtering and administration of requests and related responses) for the Intrusion Detection and Filtering Service.

The demonstration code for the following structure has been developed:

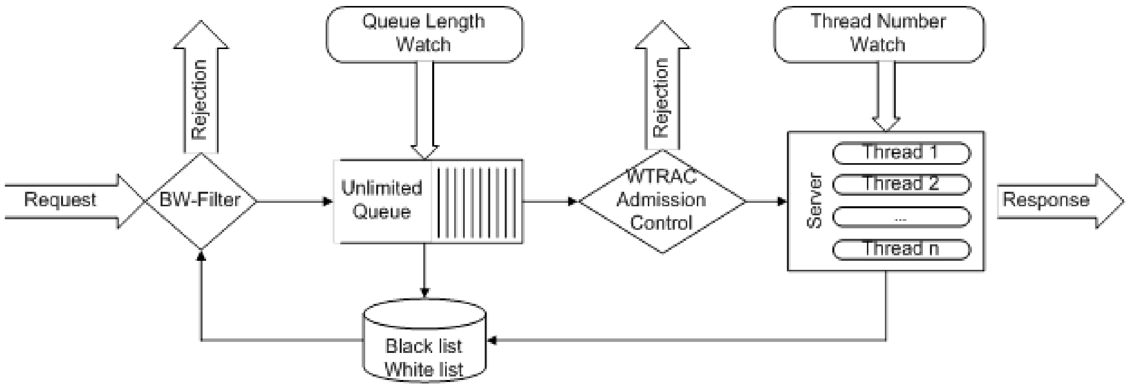


Figure 3-31: The Logic Block Structure of the DoS Protection Subsystem

The current code delivered implements filtering based on automatically black-listing based on queue length and server status watch.

3.3.2 Architecture Modules and Interfaces

3.3.2.1 Implemented Modules

From the modules described in D2.4 Reference System Architecture Design, chapter 6.4 Middleware, the following modules are implemented:

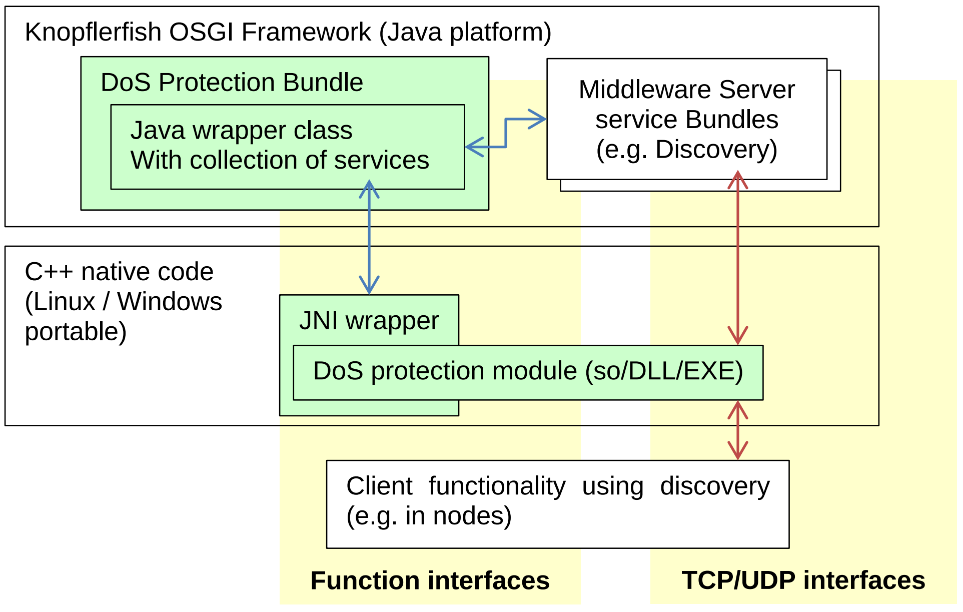

Knopflerfish OSGI Framework (Java platform)
DoS Protection Bundle
Java wrapper class With collection of services
Middleware Server service Bundles (e.g. Discovery)
C++ native code (Linux / Windows portable)
JNI wrapper
DoS protection module (so/DLL/EXE)
Client functionality using discovery (e.g. in nodes)
Function interfaces
TCP/UDP interfaces

Figure 3-32: Components of Intrusion Detection and Filtering module

On the logical level, relevant functionality of Intrusion Detection and Filtering is implemented.

On the deployment level, the deliverable consists of the source for the Intrusion Detection and Filtering module. This module is built up from the following code parts, as shown in Figure 3-32, coloured in light green.

The System Monitoring (measurement collection) functionality is not included in the preliminary version. This functionality will rely on definition of the collection of measurements as required by Overlay modules, as well as relevant Overlay functionalities to issue measurement requests and to consume measurement values collected from system components.

3.3.2.2 Interfaces

The current version of the DoS protection subsystem implements interfaces to be used for demonstration purposes. As such, the module can be considered as a TCP/UDP gateway with Java function interfaces for control.

Configuration and control is possible via several methods:

- DoS protection module can be started in stand-alone way, setting parameters in command-line and/or in configuration file.
- DoS protection module has a frontend implemented in Java as an OSGI Bundle so that it can be controlled from a Knopflerfish OSGI Framework, and other bundles may start and control DoS protection services via its function interfaces.

The Intrusion Detection and Filtering module has the following interfaces:

1. Multiple DoS protection services can be started to listen to client requests on given ports. TCP or UDP connections/datagrams will be accepted by DoS protection module from the clients. Arbitrary data received on these interfaces (subject to filtering) is be forwarded towards the server services without alteration, and response data is directed back from the server towards this interface. (Multiplicity = number of services started).
2. DoS protection module forwards filtered client requests to Server service listening on a server (TCP/UDP) port. Then, server response is received and forwarded towards clients. (Multiplicity = number of services started in 1.)
3. DoS protection subsystem itself has a control interface implemented in Java providing the following:
    3.1. OSGI bundle activator interface (Start/Stop operations for the whole subsystem)

    3.2. Collection of services, and methods to create and destroy new services

    3.3. Blacklist addition and removal, Whitelist addition and removal functions

    3.4. Functions for service control: Start and Stop, Set maximal concurrent listeners, Set queue size, Set critical load status, Set filtering mode (none, based on whitelist, based on blacklist)

    3.5. Functions to retrieve service metrics (incoming requests, outgoing requests, dropped requests, queue length, blacklist rejections, whitelist rejections)
4. For development and demonstration purposes, the DoS protection subsystem has logging implemented, where relevant events and internal status is output continuously.

Other functionality relying on Filtering and Intrusion Detection module should start a service for itself setting relevant parameters of the service (name, IP address of service, client and server ports, and protocol). Interfaces 3.X are offered to other Middleware and Overlay services for control, setup, and monitoring of the services.

The code supplied in D5.2 contains code demonstrating the usage of this service, as well as test cases for testing all major functions of the Filtering and Intrusion Detection module.

Further integration is also planned in the next phases to evaluate usage of virtual network interfaces [28], to provide unified entry points for this service in case of heterogeneous protocols used by Discovery and Composition.

3.3.2.2.1 TAP / TUN virtual network interfaces provided by the IDS

To enable transparency, to the end-devices, for accessing the overlay network we propose the usage a virtual interface that "intercepts" users' traffic and forwards it to the overlay. This interface operates transparently to any application and serves as a single gateway between end-device and the overlay. The traffic destined to the overlay provided service, instead of following the normal network path are sent to this virtual interface. TUN simulates a network layer device and it operates on layer 3 packets such as IP packets. In this approach, packets sent by an operating system via this virtual device are delivered to a user-space program that attaches itself to the device.

To implement this feature we relied on the TUN pseudo-device driver [29], while the TUN based service is implemented in the C programming language. The TUN interface can be used as a bridge for any application requires accessing the nSHIELD infrastructure. The architecture of the developed TUN service is depicted in the following Figure.

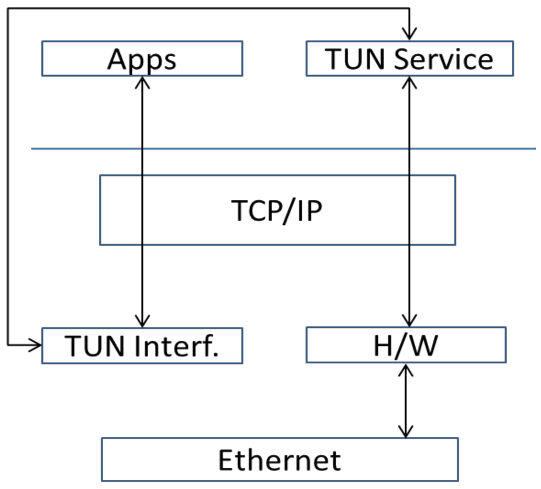


Figure 3-33: The components and functional operation of a TUN based service

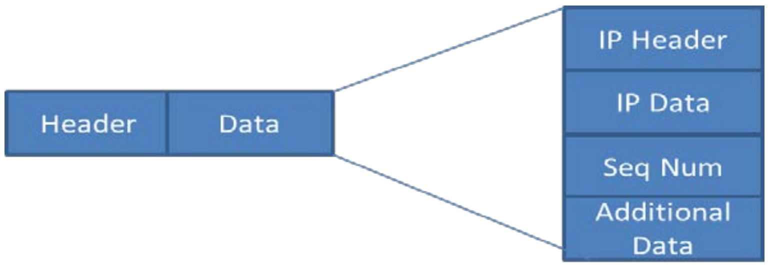


Figure 3-34: Packet structure

In this case TUN device delivers these packets to the operating system network stack thus emulating their reception from an external source. The packets that received by the interface are encapsulated in new UDP/TCP packet. The decision to use UDP instead of TCP for packet encapsulation is based on the fact that the encapsulation of TCP within TCP is not efficient [29]. The format of the new packet is illustrated in the following Figure.

3.3.2.2.2 Building an overlay using TUN/TAP Interface

In the following figure is illustrated an example where an end-user has installed the provided interface and forwards its traffic to the overlay network. In that case, it is supposed that the end-user has installed in his/her device the implemented software.

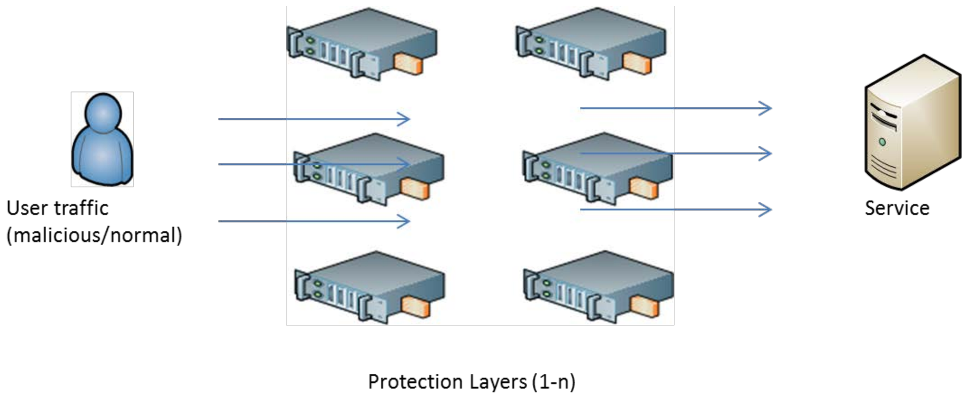


Figure 3-35: Accessing the network through the virtual interface

Particularly, the proposed architecture is consisted of

- the “client” module
- the forwarder module

<u>**Client Module**</u>

In the client side the data targeting a service instead of sent to the Ethernet interface are forwarded to the tun interface. In order to achieve this we should modify the root tables in the client using the following commands:

1. *ifconfig tun0 tunIP up*
2. *route add -host service-to-accessed dev tun0*

The first command assigns a new IP to the tun, while the second on redirect the traffic to the tun0 interface. As the tun interface delivers the original packet to the user-space we capture the packets in the application layer by creating a socket which is associated with the tun interface. This functionality is accomplished by the coded depicted in the following figure.

```
/*read from the tun*/
if(FD_ISSET(tap_fd, &rd_set))
{
     //read from tun
    len=tun_read(tun_fd,outgoing,sizeof(outgoing));
     //process the outgoing data
}
```

Figure 3-36: Example of reading data from tun interface at the client side

Forwarder Module

The Forwarder Module as its name implies is responsible to forward the received packet to a randomly chosen node of the next layer using the code illustrated in Figure 3-37.

```
len = recvfrom(sd2, buf, PDU, 0, (struct sockaddr *) &cliAddr, &cliLen);
if(len<0)
{
    printf("%s: cannot receive data \n",argv[0]);
    continue;
}
else
{
    selectedNode=chooseNode(overlayNodes);
    error= sendto(sd1, buf, len,
                  MSG_DONTWAIT,
                  (struct sockaddr *)&remoteServAddr[selectedNode],
                   sizeof(remoteServAddr[selectedNode]));
     if(error<0)
     {
        perror("cannot send data");
        close(sd1);
        exit(1);
     }
}
```

Figure 3-37: Example of forwarding data from one node to another node of the overlay

The last layer of the overlay de-encapsulates the initial packet and sends it to the service. Depending on the configuration the responses of the service can be routed either through the overlay or can be sent directly to the end-user.

```
/*RAW Socket Setup, for raw packet forwarding */
if ((sd1 = socket(PF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)
{
      perror("Forwarding socket Error");
      exit(1);
}
tmp = 1;
setsockopt(sd1, IPPROTO_RAW, IP_HDRINCL, &tmp, sizeof(tmp));
len = recvfrom(sd2, buf, PDU, 0, (struct sockaddr *) &cliAddr, &cliLen);

if(len<0)
{
    printf("%s: cannot receive data \n",argv[0]);
    continue;
}
else
{
    error = sendto (sd1, buf, len, 0, (struct sockaddr *) &sin, sizeof(sin));
    if(error<0)
    {
        printf("%s: cannot send data \n",argv[0]);
        close(sd1);
        exit(1);
    }
}
```

Figure 3-38: Example of forwarding the raw IP data to the real service

3.3.2.2.3 Detection Module

The detection module incorporates the functionality which illustrated in Figure 3-32 in order to eliminate the attempts of a node participating in the nSHIELD architecture to accomplish a DoS. To do this each node is authenticated first to the protection service and then get access to the actual service. This can be achieved by using either symmetric or asymmetric keys depending on the computation “capabilities” of the end-user and the other security parameters that will be used in the nSHIELD architecture. If the protection service authenticates the “end-user” successfully provides to him an authentication ticket which should incorporate in all the requests sent to the service by relying on the structure depicted in Figure 3-32. In the case which the ticket is missing the request immediately rejected otherwise the request is forwarded to the core of the detection module of DoS engine. The core monitors end-users’ flows as entering the system and assess as malicious if exceeding a pre-defined threshold. A sample code of this module is illustrated in Figure 3-39.

```
int initFlow()
{
        int i;
        int flow_size=RECORD_NUM;

        for(i=0;i<flow_size;i++)
       {
                flows[i].counter=0;
                flows[i].timer=0;
                flows[i].punishment=1;
        }
}

int setFlowCnt(int flowId)
{
        struct timeval tim;
        flows[flowId].counter=flows[flowId].counter++;
        gettimeofday(&tim,NULL);
        flows[flowId].timestamp=tim.tv_sec+(tim.tv_usec/1000000.0);

}

Void monitor (int i)
{
        double t2;
        double time_diff;
        struct timeval tim;
        struct itimerval tout_val;
        signal(SIGALRM,monitor);
        howmany += INTERVAL;

        /*reset the timer, we call this function every 5 seconds*/
        tout_val.it_interval.tv_sec = 0;
        tout_val.it_interval.tv_usec = 0;
        tout_val.it_value.tv_sec = INTERVAL; /* 5 seconds timer */
        tout_val.it_value.tv_usec = 0;
        setitimer(ITIMER_REAL, &tout_val,0);

        /*here I have to make my checks*/
        /*get the current time*/
        gettimeofday(&tim,NULL);
        t2=tim.tv_sec+(tim.tv_usec/1000000.0);
        //check all the records of the monitor
        for(i=0;i<RECORD_NUM;i++)  {
                time_diff=t2-flows[i].timestamp;
                if(time_diff>=INTERVAL && flows[i].timestamp!=0)
               {
                    if(flows[i].counter>THRESHOLD){
```

```
                                   //flow rejection
                              }
                              flows[i].timestamp=0;
                              flows[i].counter=0;
                          }
                          else {
                             if(flows[i].counter>THRESHOLD)  {
                                   //flow rejection
                             }
                          }
                 }
}
```

Figure 3-39: Sample Code Example of the Flow Control Monitor for the TUN Interface

3.3.2.2.4 Interface to IDS

All the overlay modules (client and forwarder) can interact with IDS by forwarding the traffic to it or by integrating the IDS functionality in it as illustrated in Figure 3-40 and Figure 3-41. This architecture can consist of a number of overlay-nodes. This way the protection service will not rely on a single point of failure, which is considered crucial for the nSHIELD architecture.

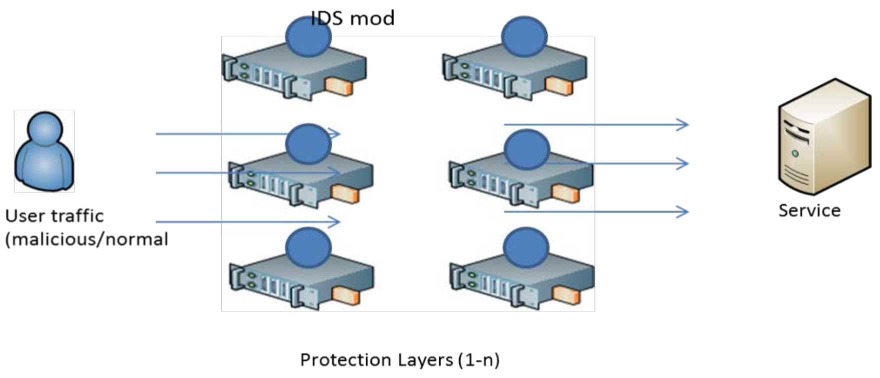


Figure 3-40: Embedding the IDS Functionality in every forwarder

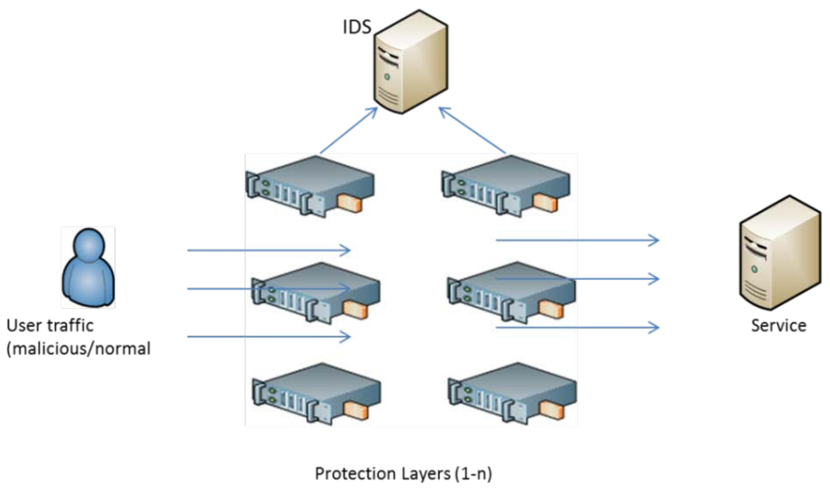


Figure 3-41: A centralized architecture for detecting malicious activity

3.3.3 Metrics

The current preliminary version of the DoS Protection module implements output of values relevant to SPD metrics values as described in D5.2 Preliminary SPD Metrics Specification, section 5.3 Middleware SPD metrics. For the following integration phases, metrics values are available for being forwarded to relevant modules as SPD metrics.

The actual values reported in the current version, and the relevant SPD metrics values are shown in the following table. All values are currently ordinal values in the range 0...N, where N depends on the configuration settings of the module.

Table 3-3: Intrusion Detection System metrics values

Value	Derived SPD metrics
NumIncomingRequests	Discovery frequency
NumOutgoingResponses	Discovery Service Statistics, Composition Service Statistics
NumWTRACAllowed	Discovery Service Statistics, Composition Service Statistics
NumWTRACRejected	Failed Discovery Requests Failed Composition Requests
NumServerSent	Discovery Service Statistics, Composition Service Statistics
InServer	Discovery Service Statistics, Composition Service Statistics
Queue Length	Discovery Service Statistics, Composition Service Statistics
Rejected by Blacklist	Rejected Discovery Requests Rejected Composition Requests
Blacklist entries	Blacklist/whitelist additions and removals for DOS protection

3.4 Adaptation of legacy systems

In this section the Adaptation of Legacy Systems will be introduced, by analysing again the SLP protocol on a different point of view, i.e. the implementation in a legacy system.

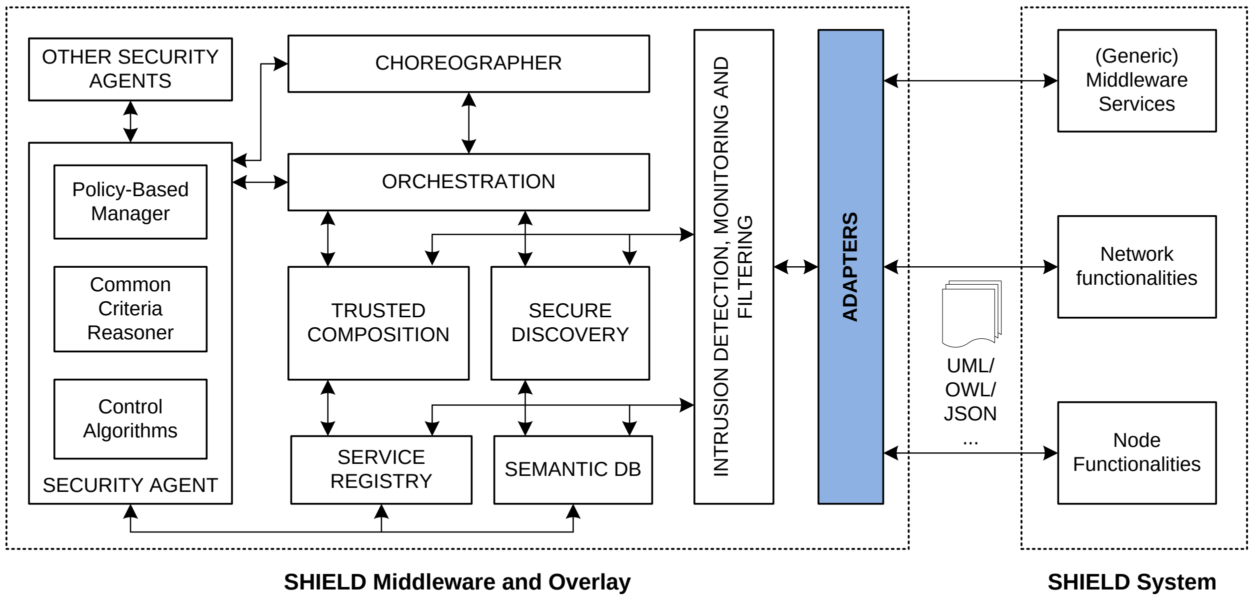


Figure 3-42: generic SHIELD adapters

3.4.1 Description

The nSHIELD architecture should be also generic enough in order to allow participation of legacy embedded systems not capable to support the nSHIELD SPD modules.

In order to allow 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).

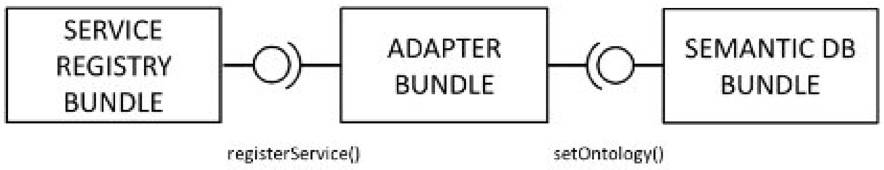


Figure 3-43: generic SHIELD adapter interfaces

In Figure 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;

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

Therefore nSHIELD can be regarded as a network consisting of nSHIELD and Legacy embedded devices having a physical architecture similar to the one depicted in Figure 3-44. The L-ESDs since they do not understand nSHIELD middleware services they need a gateway nSHIELD device in order to participate in the nSHIELD system.

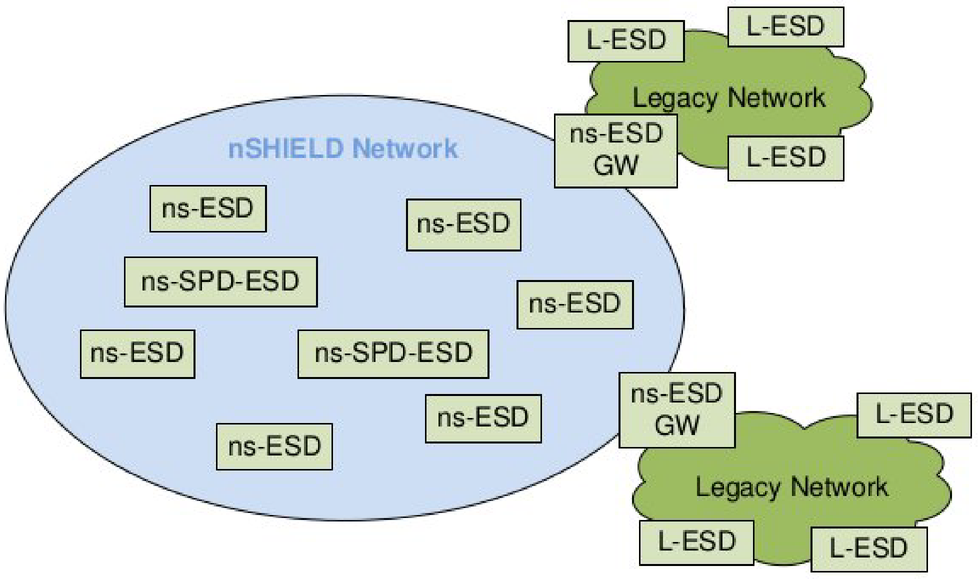


Figure 3-44: nSHIELD architecture

In general the nS-ESD GW at this level is responsible for providing proxy or adapter services for L-ESDs. The “Legacy Network or Middleware” cloud abstracts the physical and/or logical communication capabilities between the nS-ESD GW and the various L-ESDs. This node may contain additional technology dependant components which may either translate nSHIELD middleware requests to the ones that a legacy middleware can process or adapt proprietary network interfaces so that interaction within the nSHIELD network is possible for non nSHIELD compliant devices.

So we are developing SW adapters based on SLP (Service Location Protocol) [30], [31] . SLP implementation provides a framework, [32], to allow networking applications to discover the existence, location, and configuration of networked services in enterprise networks.

3.4.2 SW adapters based on SLP

3.4.2.1 AGENTS

In SLP an agent is a software entity that processes SLP protocol messages. There are three types of SLP agents:

3.4.2.1.1 User Agent (UA)

The SLP *User Agent* is a software entity that is looking for the location of one or more services. Usually implemented (at least partially), as a library to which client applications link, it provides client applications with a simple interface for accessing SLP registered service information.

3.4.2.1.2 Service Agent (SA)

The SLP *Service Agent* is a software entity that advertises the location of one or more services. SLP advertisement is designed to be both scalable and effective, minimizing the use of network bandwidth through the use of targeted multi-cast messages, and uni-cast responses to queries.

3.4.2.1.3 Directory Agent (DA)

The SLP *Directory Agent* is a software entity that acts as a centralized repository for service location information. Both Service Agents and User Agents make it a priority to discover available Directory Agents, as using a Directory Agent minimizes the amount of multi-cast messages sent by the protocol on the network.

3.4.2.2 MESSAGES

In order to be able to provide a "framework" for service location, SLP agents communicate with each other using eleven different types of messages. The dialog between agents is usually limited to very simple exchanges of request and reply messages.

3.4.2.2.1 Service Request (SrvRqst)

Message sent by UA's to SA's and DA's to request the location of a service.

3.4.2.2.2 Service Reply (SrvRply)

Message sent by SA's and DA's in response to a SrvRqst message. The SrvRply contains the URL of the requested service.

3.4.2.2.3 Service Registration (SrvReg)

Message sent by SA's to DA's containing information about a service that is available.

3.4.2.2.4 Service Deregister (SrvDeReg)

Message sent by SA's to inform DA's that a service is no longer available.

3.4.2.2.5 Service Acknowledge (SrvAck)

A generic acknowledgment that is sent by DA's to SA's in response to SrvReg and SrvDeReg messages.

3.4.2.2.6 Attribute Request (AttrRqst)

Message sent by UA's to request the attributes of a service.

3.4.2.2.7 Attribute Reply (AttrRply)

Message sent by SA's and DA's in response to a AttrRqst. The AttrRply contains the list of attributes that were requested.

3.4.2.2.8 Service Type Request (SrvTypeRqst)

Message sent by UA's to SA's and DA's requesting the types of services that are available.

3.4.2.2.9 Service Type Reply (SrvTypeRply)

Message by SA's and DA's in response to a SrvTypeRqst. The SrvTypeRply contains a list of requested service types.

3.4.2.2.10 DA Advertisement (DAAdvert)

Message sent by DA's to let SA's and UA's know where they are.

3.4.2.2.11 SA Advertisement (SAAdvert)

Message sent by SA's to let UA's know where they are.

3.4.2.3 SECURITY

SLPv2 has been designed to be a secure protocol. When properly implemented, SLPv2 can ensure integrity and authenticity of data being transmitted between SLP agents. See RF2608 section 9.2, [33] for more information.

3.4.2.4 SCALABILITY

SLPv2 was designed to be a scalable solution for enterprise service location. It is not intended to be a solution for the global Internet. However, as an enterprise solution, SLP can be configured to use "scopes" (see RFC 2608, section 11 [33]) and SLP Directory Agents in ways that should allow it to scale well in very large networks. More concrete evidence of SLPv2 scalability will become available when SLP is more widely used.

3.4.2.5 IMPLEMENTATIONS

The following is a list of known SLP implementations which we examine for developing the SW (adapters, interfaces, enablers), to make legacy devices interwork transparently with the enhanced capabilities provided by the SHIELD approach.

OpenSLP

It is an OpenSource project that aims to provide a full SLPv2 implementation [34]. Today, most Linux distributions either pre-install OpenSLP, or make it available to the user via the distribution's package management software.

JSLP

jSLP [35] can either operate stand-alone (in peer-to-peer mode), or with a dedicated SLP Directory Agent in the network. In the first case, jSLP uses multicast convergence to query the local subnet for peers that offer requested services. In the latter case, the central Directory Agent maintains all registered services and answers requests.

The SLP protocol is self-adaptive in the sense that whenever a Directory Agent is present (and matches the scope), it is exclusively used. Otherwise, multicast convergence is used. jSLP fully complies to this behaviour, [36].

jSLP-OSGi

jSLP OSGi is designed to enable SLP service discovery on OSGi platforms [37]. Bundles can get Locator and Advertiser instances to find other services in the Network. The OSGi version has a smaller footprint than the jSLP standalone version because it uses the `frameworkFilter` and instead of using commons-logging, it makes use of the OSGi org.osgi.service.log logger. The OSGi version of jSLP registers the `ServiceLocationManager` as a `ServiceFactory` for `ch.ethz.iks.slp.Advertiser` and `ch.ethz.iks.slp.Locator` services.

Since OSGi does not provide any general way to parameterize `ServiceFactories`, both `Locator` and `Advertiser` have an additional `void setLocale(Locale locale)` method to set the locale after having retrieved the service object.

3.4.2.6 IMPLEMANTATION EXAMPLES

There are some different approaches between these implementations which we are working and we try to find the most suitable platform according to SHIELD approach,

3.4.2.6.1 OpenSLP

OpenSLP is able to statically register legacy services (applications that were not compiled to use the SLP library). To accommodate this need RFC 2614, [38] specifies syntax for a registration file that is read by the OpenSLP daemon (slpd). All of the registrations from the registration file are maintained by SLP and will remain registered as long as slpd is alive. `Slpd reads the slp.reg` file on start-up and re-reads it whenever the SIGHUP signal is received.

3.4.2.6.2 Syntax

The registration file format is pretty easy to understand [39]. Each registration consists of several lines with the following format:

```
#FILE

#comment
;comment
service-url,language-tag,lifetime,[service-type]<newline>
"scopes="[scope-list]<newline>
[attrid]"="val1<newline>
[attrid]"="val1,val2,val3<newline>
<newline>

#EOF
```

3.4.2.6.3 jSLP

Register a service

The following example shows how to register a service with jSLP:

```
//CODE

// get Advertiser instance
Advertiser advertiser = ServiceLocationManager.getAdvertiser(new Locale("en"));

// the service has lifetime 60, that means it will only persist for one minute
ServiceURL myService = new ServiceURL("service:test:myService://my.host.ch",60);

// some attributes for the service
Hashtable attributes = new Hashtable();
attributes.put("persistent", Boolean.TRUE);
attributes.put("cool", "yes");
attributes.put("max-connections", new Integer(5));

advertiser.register(myService, attributes);

//END OF CODE
```

Locate a service

The next example shows how to locate a service in the network:

```
//CODE

// get Locator instance
Locator locator = ServiceLocationManager.getLocator(new Locale("en"));
```

```
// find all services of type "test" that have attribute "cool=yes"
ServiceLocationEnumeration sle = locator.findServices(new
ServiceType("service:test"), null, "(cool=yes)");

// iterate over the results
while (sle.hasMoreElements())
{
    ServiceURL foundService = (ServiceURL) sle.nextElement();
    System.out.println(foundService);
}

//END OF CODE
```

3.4.2.6.4 jSLP-OSGi

jSLP OSGi registers `ch.ethz.iks.slp.ServiceLocationManager` as `ServiceFactory`. The registered services are `ch.ethz.iks.slp.Advertiser` and `ch.ethz.iks.slp.Locator` and every bundle requesting one of the services will get their own instance. Since requests `forServiceReferences` cannot pass parameters to the `ServiceFactory`, the `Advertiser` and `Locator` instances will be created with the empty default `Locale` and both classes have a setter method `.setLocale(Locale locale)` to change the locale at runtime (this differs from the jSLP standalone version).

The following example shows how to get `Advertiser` and `Locator` instances and use them:

```
/////CODE/////

public class SLPTestBundle implements BundleActivator
{
  public void start(BundleContext context) throws Exception
  {
     ServiceReference advRef =
            context.getServiceReference("ch.ethz.iks.slp.Advertiser");
     ServiceReference locRef =
            context.getServiceReference("ch.ethz.iks.slp.Locator");

     if (advRef != null)
     {
       System.out.println("Got reference for Advertiser");
       Advertiser advertiser = (Advertiser)context.getService(advRef);
       advertiser.register(new
           ServiceURL("service:osgi:test://192.168.24.118", 20),null);
     }

     if (locRef != null)
     {
        System.out.println("Got reference for Locator");
        Locator locator = (Locator) context.getService(locRef);

       ServiceLocationEnumeration slenum = locator.findServices(new
                 ServiceType("service:osgi"), null, null);
       System.out.println("RESULT:");

       while (slenum.hasMoreElements())
       {
            System.out.println(slenum.nextElement());
       }
    }
 }
```

```
  public void stop(BundleContext context) throws Exception
  {

  }

 }
/////END OF CODE/////
```

Since the OSGi is widely preferred so far in this project, the implementation of jslp- OSGi and R-OSGi for registering and getting services is more suitable for our purpose.

OSGi is a 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 OSGi bundle can theoretically be installed in any valid OSGi container. There are many OSGi container implementations (Equinox, Felix, Knopflerfish, and ProSyst, among others). Our software is developed under Knoperflish implementation.

Some bundles must be installed in the L-ESD and Ns-ESD GW to accomplish the use of nSHIELD services by the legacy systems.

- The “R-OSGi” service runs as a bundle and will be used by bundles in nS-ESD GW for advertising nSHIELD services to Legacy Network which consists of L-ESD.
- The “jslp-osgi”, “R-OSGi SLP Service Discovery” and “R-OSGi” services will be used by bundles in L-ESD for locating the nSHIELD services in Ns-ESD GW.

Similar to RFC 2614, jSLP separates the UA and SA functionalities. According to this the UA functionality must be implemented in the Legacy devices for discovering the nSHIELD remote services and the SA functionality must be implemented in the nSHIELD GW device to advertise the nSHIELD Services.

The nSHIELD GW nodes-server side- must register their services to R-OSGi service for being accessible from the legacy nodes in the network. The have to implement it by the following ways:

1. Registration of service inside their own bundles
2. Registration of service by another bundle.

For example, the HttpService have to be registered by itself in HttpBundle or by another Bundle.

3.4.3 Registering a service for remote access (service provider side)

```
public class Activator implements BundleActivator
{
  public void start(BundleContext bundleContext)
  {

    System.out.println("Hello started.");

    //Map properties = new HashMap(0);
    Dictionary<String,Boolean> properties = new Hashtable();

    // this is the hint for R-OSGi that the service
    // ought to be made available for remote access
    properties.put(RemoteOSGiService.R_OSGi_REGISTRATION, Boolean.TRUE);
    bundleContext.registerService(Http.class.getName(), new HttpImpl(),
               properties);
  }
  public void stop(BundleContext bundleContext)
  {
    System.out.println("Hello stopped.");
  }
}
```

Now on the other side, the Legacy nodes -client side-have to get the services that are advertised by R-OSGi in the nSHIELD-GW .They have to implement it by creating bundles for locating and getting remote access to the services.

3.4.4 Connect to a remote peer and get the service (service consumer side)

```
public class Activator implements BundleActivator
{
  /* (non-Javadoc)
   * @see
org.osgi.framework.BundleActivator#start(org.osgi.framework.BundleContext)
   */
  public void start(BundleContext context) throws Exception
  {
      // get the RemoteOSGiService
      final ServiceReference sref =
           context.getServiceReference(RemoteOSGiService.class.getName());

      if (sref == null)
      {
           throw new BundleException("No R-OSGi found");
      }

      RemoteOSGiService remote = (RemoteOSGiService)
                                  context.getService(sref);
      // connect
      remote.connect(new URI("r-osgi://150.xxx.xxx.xxx:9278"));
      final RemoteServiceReference[] srefs =
             remote.getRemoteServiceReferences(new URI("r-
             osgi://150.xxx.xxx.xxx:9278"), Http.class.getName(), null );
      Http  hi = (Http) remote.getRemoteService(srefs[0]);
  }

  /* (non-Javadoc)
   * @see
org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
   */
  public void stop(BundleContext context) throws Exception
  {
  }
}
```

3.5 SHIELD middleware protection profile definition & certification

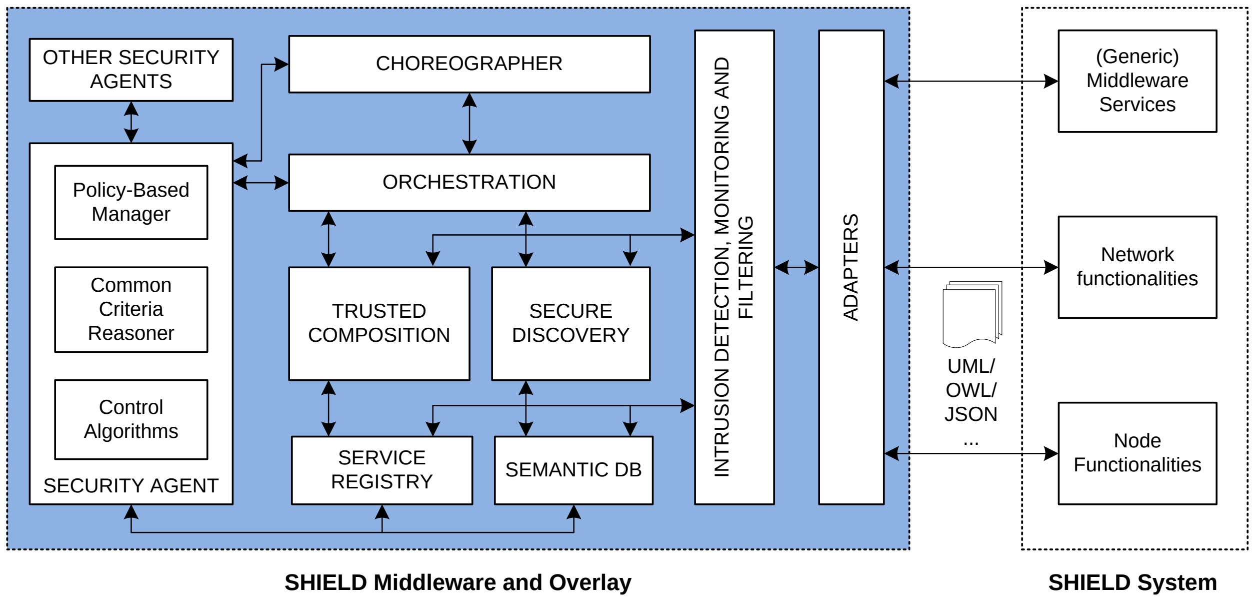


Figure 3-45: SHIELD middleware protection profile

3.5.1 Embedded systems security

An embedded system is a computer system designed to perform a dedicated or narrow range of functions with a minimal user intervention, it involves computation that is subject to several challenging traditional system design constraints.

The physical constraints arise through the two ways that computational processes interact with the physical world: reaction to a physical environment and execution on a physical platform.

a. Common reaction constraints specify deadlines, throughput, and jitter and originate from behavioural requirements.

b. Common execution constraints bound available processor speeds, power, and originate from implementation choices.

Control theory deals with reaction constraints; computer engineering deals with execution constraints.

Today embedded systems are increasingly permeating our lives, From smart buildings to automated highways, the opportunities seem unlimited but meantime the requests for security, privacy and dependability for these systems are becoming more urgent and although security issues are nothing new for embedded systems they still remain an open question and could prove a more difficult long-term problem than security does today for desktop and enterprise computing. In the modern application scenarios always more embedded systems are connected to the Internet and the potential damages from potential vulnerabilities scale up dramatically. Internet connections expose applications to intrusions and malicious attacks. Unfortunately, security techniques developed for enterprise and desktop computing might not satisfy embedded application requirements this because for embedded systems we have to take into account:

Cost sensitivity

Embedded systems are often highly cost sensitive - even five cents can make a big difference when building millions of units per year. For this reason, most CPUs manufactured worldwide use 4- and 8-bit processors, which have limited room for security overhead. Many 8-bit microcontrollers, for example, can't store a big cryptographic key. This can make best practices from the enterprise world too expensive to be practical in embedded applications. Cutting corners on security to reduce hardware costs can give a competitor a market advantage for price-sensitive products. And if there is no quantitative measure of security before a product is deployed, who is to say how much to spend on it?

Interactive matters

Many embedded systems interact with the real world. A security breach thus can result in physical side effects, including property damage, personal injury, and even death. Backing out financial transactions can repair some enterprise security breaches, but reversing a car crash isn't possible.

Energy constrains

Embedded systems often have significant energy constraints, and many are battery powered. Some embedded systems can get a fresh battery charge daily, but others must last months or years on a single battery. By seeking to drain the battery, an attacker can cause system failure even when breaking into the system is impossible. This vulnerability is critical, for example, in battery-powered devices that use power-hungry wireless communication.

Development environment

Many embedded systems are created by small development teams or even lone engineers. Organizations that write only a few kilobytes of code per year usually can't afford a security specialist and often don't realize they need one. However, even seemingly trivial programs may need to provide some level of security assurance.

3.5.2 What is a Protection Profile?

In order to address the above security, privacy and dependability (SPD) open issues, the nSHIELD project aims at addressing SPD in the context of ESs as "built in" functionalities, proposing and perceiving with this strategy the first step towards SPD certification for future ESs.

Editing a protection profile is a first step to define a security problem definition and security objectives for embedded systems, but what is a protection profile?

A protection profile (PP) is a Common Criteria[1] (CC) term for defining an implementation-independent set of security requirements and objectives for a category of products, which meet similar consumer needs for IT security. Examples are PP for application-level firewall and intrusion detection system. PP answers the question of "what I want or need" from the point of view of various parties. It could be written by a user group to specify their IT security needs. It could also be used as a guideline to assist them in procuring the right product or systems that suits best in their environment. Vendors who wish to address their customers' requirements formally could also write PP. In this case, the vendors would work closely with their key customers to understand their IT security requirements to be translated into a PP. A government can translate specific security requirements through a PP. This usually is to address the requirements for a class of security products like firewalls and to set a standard for the particular product type.

[1] *Common Criteria is a standard for security specifications and evaluation, ISO15408.*

3.5.3 Why a protection profile?

Our purpose of writing a PP in the nSHIELD project is to define the security requirements for nSHIELD complaint embedded system product. It does so by providing a framework that describes the product security environment, and objectives from which the requirements could be derived. To establish the security environment, the product must be first identified and described. The threats, organizational policies and assumptions from the product security environment become an input to determining the product security objectives. Having determined the product security objectives, the product security requirements can be drawn up using CC Part 2 and Part 3. The requirements and objectives shall be traceable to the inputs and this shall be demonstrated in the rationale statements as evidence that the PP is complete, coherent and consistent internally.

We want to follow United States consolidated experience where vendors who wish to have their products eligible as Commercial Solution for Classified (CSfC) components of a composed, layered Information Assurance (IA) solution must build their products in accordance with the applicable U.S. Government Protection Profile(s) and submit their products using the Common Criteria Process.

The nSHIELD project has the ambitious to be a commercial standard for Security, Privacy and Dependability regarding embedded systems and writing a PP for SPD requirements is the first step to establish a European standard, in the same manner the protection profile prEN 14169-1 is established by Technical Committee CEN/TC 224 as a European standard for products to create electronic signatures. It fulfils requirements of directive2 1999/93/ec of the European parliament and of the council of 13 December 1999 on *a community framework for electronic signatures*.

4 SHIELD policy based management and access control

In this section we analyse two different, yet both policy-based, mechanisms adopted by nSHIELD architecture, namely Policy-Based Management (PBM) of SPD functionalities and Policy-Based Access Control (PBAC) of nSHIELD resources. While PBM of SPD functionalities is used so that the nSHIELD system configuration can adapt dynamically and autonomously to changing conditions and maintain an acceptable security level, PBAC is used for managing access requests to resources, based on a predefined set of rules and policies.

4.1 SHIELD policy based access control (PBAC)

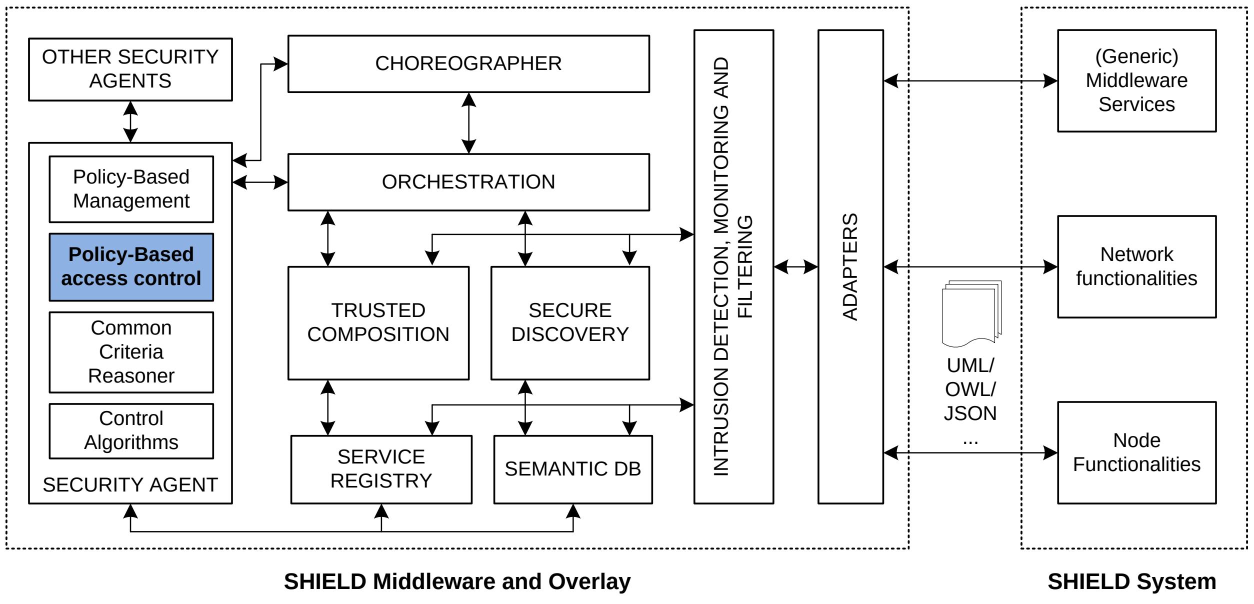


Figure 4-1: SHIELD Policy Based Management and Access Control

4.1.1 Description

The solution adopted for secure policy-based access control is based on eXtensible Access control Markup Language (XACML) policies. XACML is an XML-based general-purpose access control policy language used for representing authorisation and entitlement policies for managing access to resources. However, it is also an access control decision request/response language. As such, it can be used to convey policy requirements in a unified and unambiguous manner, hence interoperable and secure, if appropriately deployed.

The above fit well into the model of a network of heterogeneous embedded systems where access to resources is provided by nodes as a service, and into the management architecture developed by IETF Policy Framework. This typical policy based access control architecture combined with XACML, is mapped to a Service Oriented Architecture (SOA) network of nodes to provide protected access to their distributed resources.

4.1.2 Architecture Modules and Interfaces

XACML is designed to accommodate the policy management architecture which consists of the following modules:

- **Policy Enforcement Point (PEP)**: The system entity that performs access control, by making decision requests and enforcing authorization decisions. A PEP’s sub-module is the Obligation Handling (OH) module which is responsible for handling the operations specified in a rule, policy, or policy set.
- **Policy Administration Point (PAP)**: The system entity that creates a policy or policy set.
- **Policy Decision Point (PDP)**: The system entity that evaluates applicable policy and renders an authorization decision.
- **Policy Information Point (PIP)**: The system entity that acts as a source of attribute values.
- **Context Handler (CH)**: It orchestrates the communication among the stakeholders, converts, if necessary, messages between their native forms and the XACML canonical form, and collects all necessary information for the PDP

The architecture modules and their interconnections are depicted in Figure 4-2.

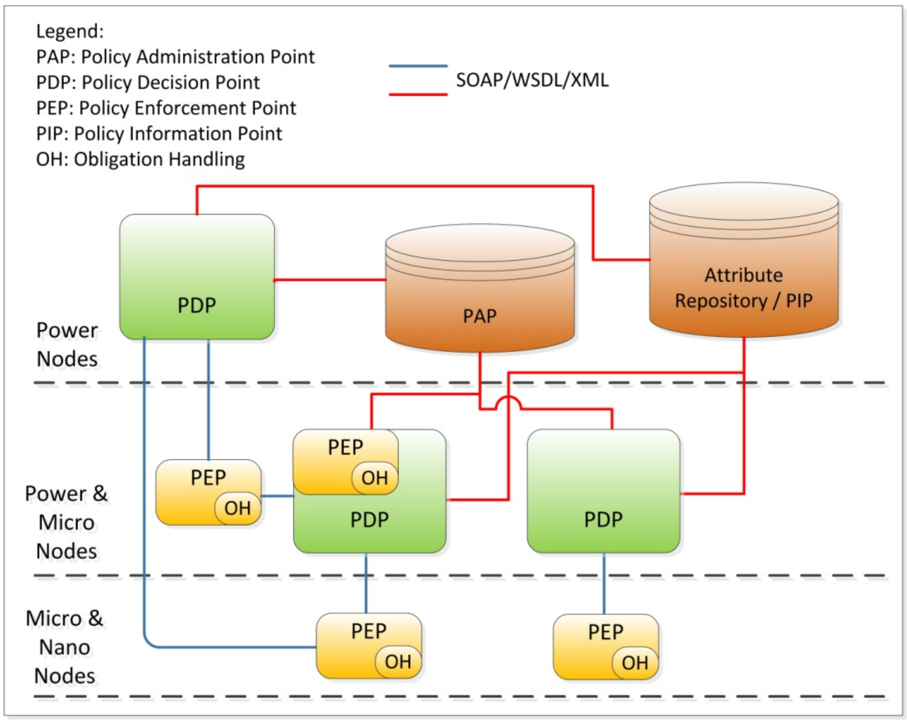

Legend:
PAP: Policy Administration Point
PDP: Policy Decision Point
PEP: Policy Enforcement Point
PIP: Policy Information Point
OH: Obligation Handling
SOAP/WSDL/XML
PDP
PAP
Attribute Repository / PIP
Power Nodes
Power & Micro Nodes
Micro & Nano Nodes
PEP
OH
PDP
PEP
OH
PDP
PEP
OH
PEP
OH

Figure 4-2: Policy based Architecture

In a typical data flow model, authorization requests for accessing nodes’ resources are forwarded to PDPs, through the context handler which is responsible for orchestrating the communications and collecting the required attributes from the PIP. The PDP evaluates the request against the policies and rules set and provided by the PAP, and issue an authorization decision which the PEP has to enforce together with some (optional) obligations and/or advices. In this process it is important not to disregard the fact that a PEP is not expected to have the capacity to support the context handler functionality, which should be offloaded to a PDP or another infrastructure component, such as a base station in a WSN architecture.

According to the proposed architecture, different approaches can be adopted for the different types of nodes met in heterogeneous systems, with diversified capabilities. The chosen architecture consists of the following components.

1) Power Nodes running both PEP and PDP functionality, serving requests originating from other nodes or for its own needs. This role is assumed by a BeagleBoard component. The PDP interacts with the PIP/PAP that handles policies and attributes necessary for making decisions. Note that although a Power node will have all the aforementioned functionality it might not assume both roles (PEP/PDP) but rather choose to act either as a PEP or a PDP.
2) Micro or Nano nodes acting only as PEPs, hence enforcing authorization decisions rendered by a PDP (Power Node). Nano nodes, such as a node running Contiki or TinyOS will not perform anything more than the PEP functionality. Micro nodes on the other hand, such as a BeagleBone running a Linux distribution, might have the capacity to accommodate the PDP functionality depending on the available resources and the environment that will be deployed.

Regarding communications protocols chosen for the exchange of the policy related messages, there is a variety of protocols proposed with different properties and characteristics to satisfy diversified environments needs. 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. Resource-constrained nodes participating in a SOA should typically avoid connection overhead caused by expensive protocols such as TCP, although this option is at the expense of reliability which has to be provided by other layers protocols. Therefore, the choice made was to use SOAP over UDP, being in line with OASIS which characterized this solution as the natural choice [23]. SOAP over UDP specifies the way SOAP envelopes are carried in user datagrams while respecting the requirements for UDP messages being received in one chunk, which demands valuable resources on small. Lower layers issues are also being considered, such as the underlying communication protocols, e.g. whether the communication would take place over 6lowpan.

Emphasis is also given on the protection of the exchanged policy messages against unauthorized disclosure, modification, and masquerade attacks. Among the available solutions the choice made was to protect the messages at the network layer, by using an IPSEC-based variant for resource constrained environments. Details of this approach are provided in D4.3.

4.1.3 Implementation details

4.1.3.1 Device classification

The policy-based management prototype will utilize the technologies listed below.

Please note that there is no explicit matching of the Policy Enforcement and Policy Decision points (i.e. PEP, PDP) to devices since the framework is flexible in this regard and exact assignment will depend on application scenario. Moreover, descriptions below are restricted to features pertaining to the proposed framework and do not include additional functions that may be running on the same devices (e.g. security agents, anonymizer and other nSHIELD services).

4.1.3.1.1 Nano nodes

Small devices with limited resources both in terms of hardware and software; typically sensor motes. Battery powered.

4.1.3.1.1.1 Role

- Devices running a DPWS service.

4.1.3.1.1.2 Underlying technologies

- Operating system: Contiki
- Network: 802.15.4/6LoWPAN
- DPWS service provision: uDPWS stack (C)

4.1.3.1.1.3 Prototype platforms

- Zolertia Z1 motes [40]

Figure 4-3: Zolertia Z1

- Crossbow Technology IRIS motes [41]

Figure 4-4: Crossbow Technology IRIS

4.1.3.1.2 Micro nodes

Devices richer than the nano nodes in terms of hardware and software resources, network access capabilities, mobility, interface etc. Considering the computing power, these nodes remain confined in the class of embedded computers. Typically battery powered.

4.1.3.1.2.1 Role

- DPWS client and server (i.e. DPWS peer).

4.1.3.1.2.2 Underlying technologies

- Operating system: A lightweight Linux distribution
- Network: 802.15.4/6LoWPAN
- DPWS functionality: WS4D-gSOAP (C)

4.1.3.1.2.3 Prototype platforms

- BeagleBone [42]

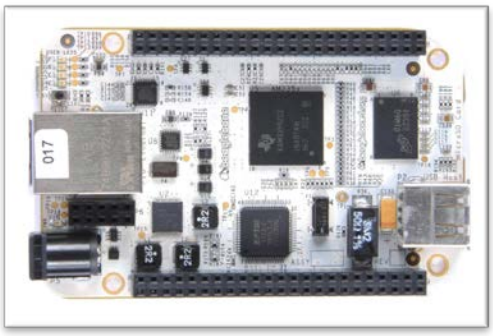


Figure 4-5: BeagleBone

4.1.3.1.3 Power nodes

Devices that maintain the characteristics of embedded systems while offering relatively high performance in terms of computing power. This class of nodes represents, in the pervasive system, the first level of massive data elaboration, with the peculiarity that the computing power is provided directly on the field. Typically mains powered.

4.1.3.1.3.1 Role

- DPWS client and server (i.e. DPWS peers).
- Responsible for interfacing with OSGi (Knopflerfish) framework.
- Bridge between 802.15.4/6LoWPAN and IPv4/IPv6 networks
- Policy Administration Point
- Policy Information Point - Attributes repository

4.1.3.1.3.2 Underlying technologies

- Operating system: A lightweight Linux distribution with desktop environment
- Network: 802.15.4/6LoWPAN, IPv4/IPv6
- DPWS functionality: WS4D-JMEDS (Java), WS4D.Comoros (DPWS-OSGi interface)
- OSGi functionality: Knopflerfish framework

4.1.3.1.3.3 Prototype platforms

- BeagleBoard xM [43]
- BeagleBoard [44]

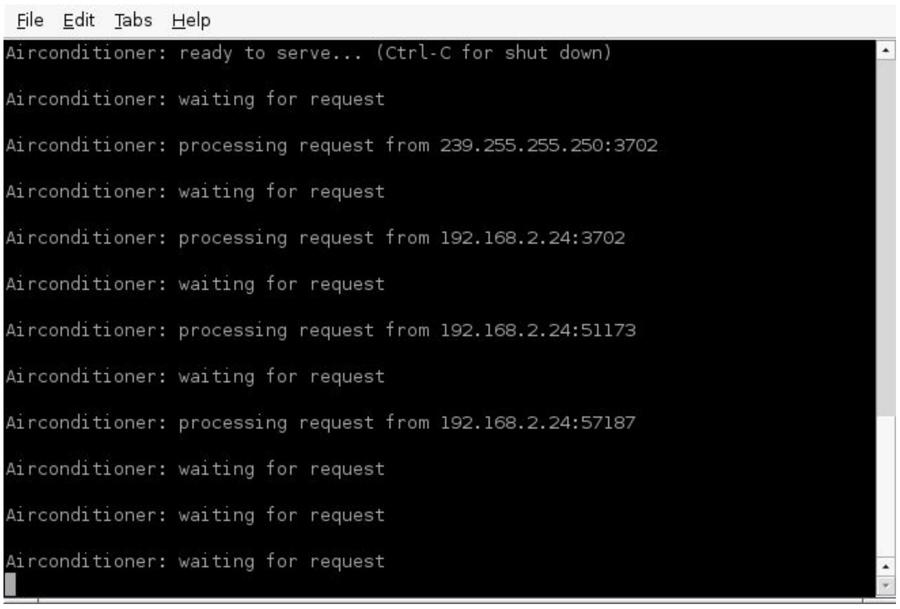
Figure 4-6: BeagleBoard-xM

Figure 4-7: BeagleBoard

4.1.3.2 Simple application

A simplified application of the proposed framework, suitable for an elementary demonstration of functionality, could entail the following:

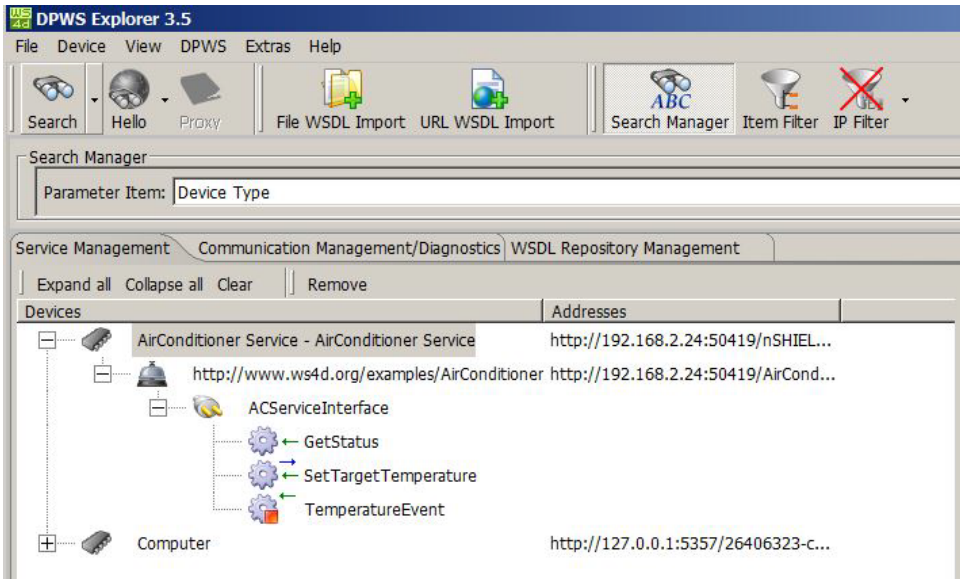
- A number of **nano nodes** (e.g. Zolertia Z1 sensor motes) are spread across a room. Each node runs a DPWS *“MOTE_X_TEMPERATURE”* service (where *X* = device’s ID). This allows clients to subscribe and get a temperature reading at set intervals (e.g. pushed to subscribed devices every 10 minutes), on request (replying to a *GetStatus* operation submitted by a client) and/or when certain events are triggered (e.g. when temperature rises above a certain threshold).
- A **micro node** (BeagleBone [42]) is embedded in each room’s smart air-conditioning appliance. It subscribes as a client to DPWS *“MOTE_X_TEMPERATURE”* services running on nano nodes present in the room. Each micro node also runs its own DPWS service, *“ROOM_Y_TEMPERATURE”* (where *Y* = room’s ID). This service reports the average of temperatures collected from sensor motes (again at set intervals, on request or when certain events are triggered) and can be considered the average room temperature. The device is also responsible for shutting down the appliance when the room temperature reaches the desired value.



```
Airconditioner: ready to serve... (Ctrl-C for shut down)

Airconditioner: waiting for request

Airconditioner: processing request from 239.255.255.250:3702

Airconditioner: waiting for request

Airconditioner: processing request from 192.168.2.24:3702

Airconditioner: waiting for request

Airconditioner: processing request from 192.168.2.24:51173

Airconditioner: waiting for request

Airconditioner: processing request from 192.168.2.24:57187

Airconditioner: waiting for request

Airconditioner: waiting for request

Airconditioner: waiting for request
```

Figure 4-8: A BeagleBoard runs an “Airconditioner” DPWS service

- A **power node** (BeagleBoard-xM [43]) is the control point of a smart home environment or of a floor of a smart office building. It subscribes as a client to all DPWS *“ROOM_Y_TEMPERATURE”* services running on micro nodes (i.e. smart air-conditioning in this application) in its range (e.g. its floor). Power node is responsible for setting the desired temperature of each room (via a *SetTargetTemperature* operation of the *“ROOM_Y_TEMPERATURE”* service*)*, shutting down devices when outside working hours or in cases of emergency etc.



Parameter Item: Device Type

Devices	Addresses
AirConditioner Service - AirConditioner Service	http://192.168.2.24:50419/nSHIEL...
http://www.ws4d.org/examples/AirConditioner	http://192.168.2.24:50419/AirCond...
ACServiceInterface	
GetStatus	
SetTargetTemperature	
TemperatureEvent	
Computer	http://127.0.0.1:5357/26406323-c...

Figure 4-9: A DPWS client (laptop computer running DPWS Explorer) discovers the Airconditioner service running on BeagleBoard

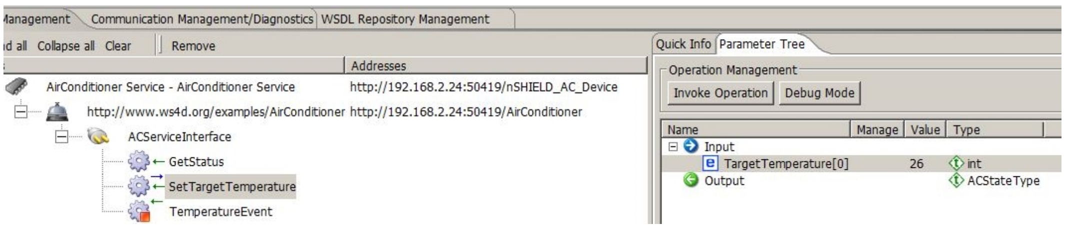
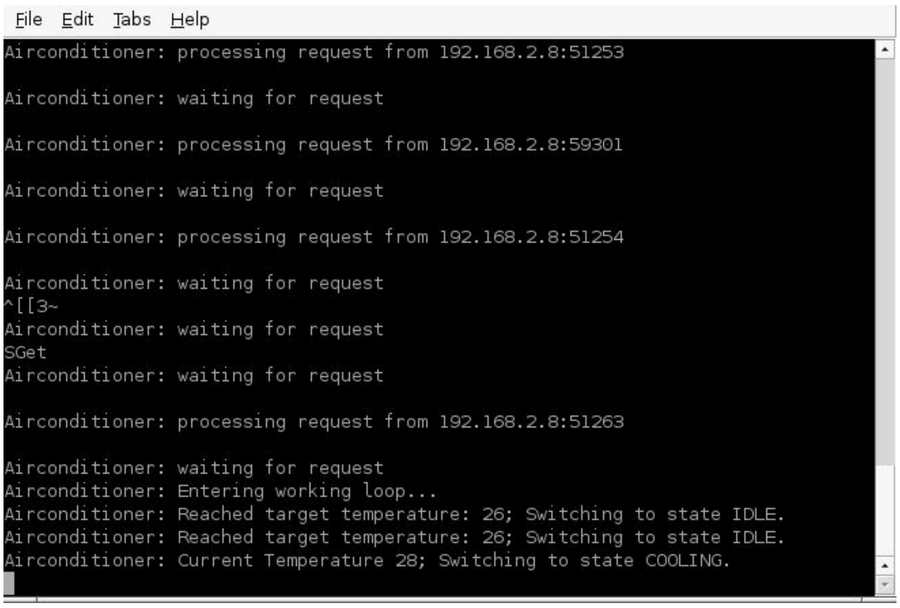


Figure 4-10: Client setting target temperature of Airconditioner service to 26 degrees



```
Airconditioner: processing request from 192.168.2.8:51253

Airconditioner: waiting for request

Airconditioner: processing request from 192.168.2.8:59301

Airconditioner: waiting for request

Airconditioner: processing request from 192.168.2.8:51254

Airconditioner: waiting for request
^[[3~
Airconditioner: waiting for request
SGet
Airconditioner: waiting for request

Airconditioner: processing request from 192.168.2.8:51263

Airconditioner: waiting for request
Airconditioner: Entering working loop...
Airconditioner: Reached target temperature: 26; Switching to state IDLE.
Airconditioner: Reached target temperature: 26; Switching to state IDLE.
Airconditioner: Current Temperature 28; Switching to state COOLING.
```

Figure 4-11: Device switching states based on target temperature set by DPWS client

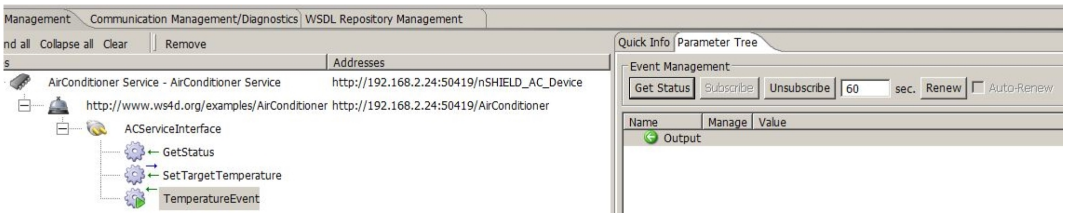
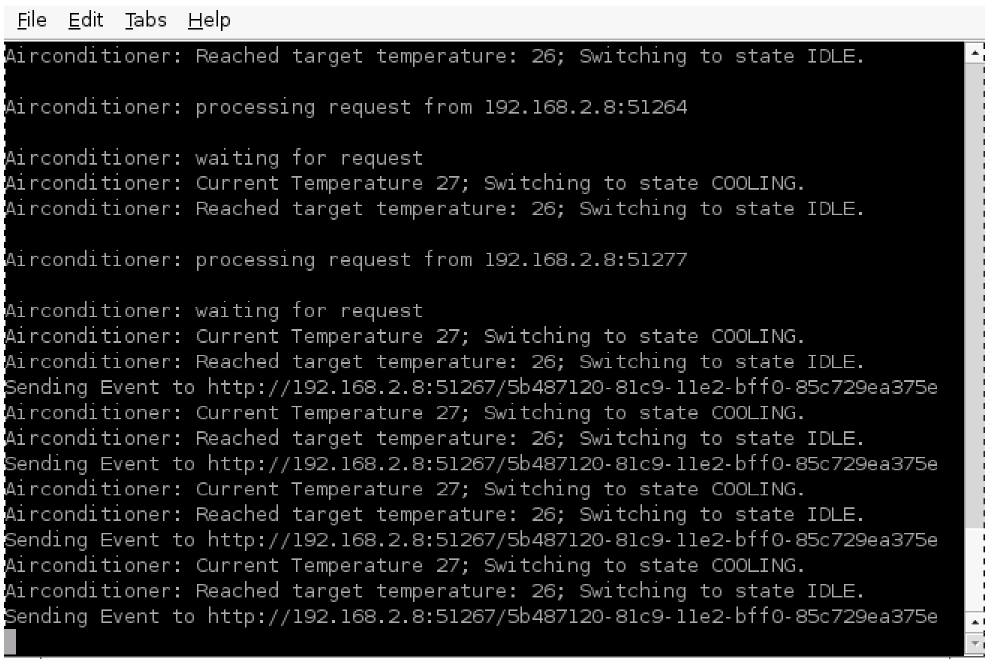

Management
Communication Management/Diagnostics
WSDL Repository Management
Collapse all
Clear
Remove
Addresses
AirConditioner Service - AirConditioner Service
http://192.168.2.24:50419/nSHIELD_AC_Device
http://www.ws4d.org/examples/AirConditioner
http://192.168.2.24:50419/AirConditioner
ACServiceInterface
GetStatus
SetTargetTemperature
TemperatureEvent
Quick Info
Parameter Tree
Event Management
Get Status
Subscribe
Unsubscribe
60
sec.
Renew
Auto-Renew
Name
Manage
Value
Output

Figure 4-12: DPWS client subscribing to TemperatureEvent operation of server (implemented using WS-Eventing)



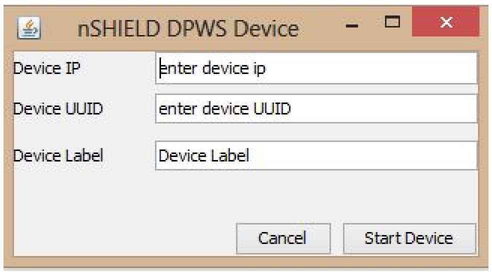
```
Airconditioner: Reached target temperature: 26; Switching to state IDLE.

Airconditioner: processing request from 192.168.2.8:51264

Airconditioner: waiting for request
Airconditioner: Current Temperature 27; Switching to state COOLING.
Airconditioner: Reached target temperature: 26; Switching to state IDLE.

Airconditioner: processing request from 192.168.2.8:51277

Airconditioner: waiting for request
Airconditioner: Current Temperature 27; Switching to state COOLING.
Airconditioner: Reached target temperature: 26; Switching to state IDLE.
Sending Event to http://192.168.2.8:51267/5b487120-81c9-11e2-bff0-85c729ea375e
Airconditioner: Current Temperature 27; Switching to state COOLING.
Airconditioner: Reached target temperature: 26; Switching to state IDLE.
Sending Event to http://192.168.2.8:51267/5b487120-81c9-11e2-bff0-85c729ea375e
Airconditioner: Current Temperature 27; Switching to state COOLING.
Airconditioner: Reached target temperature: 26; Switching to state IDLE.
Sending Event to http://192.168.2.8:51267/5b487120-81c9-11e2-bff0-85c729ea375e
Airconditioner: Current Temperature 27; Switching to state COOLING.
Airconditioner: Reached target temperature: 26; Switching to state IDLE.
Sending Event to http://192.168.2.8:51267/5b487120-81c9-11e2-bff0-85c729ea375e
```

Figure 4-13: Sending events to subscribed client

- In a typical policy-based managed data flow model, authorization requests for accessing nano and micro nodes' DPWS services and issuing commands (requesting operations) to said devices are forwarded to PDPs, through the context handler which is responsible for orchestrating the communications and collecting the required attributes from the PIP/PAP (present on power nodes). The PDP evaluates the request against the policy restrictions taken from the PAP (running on power nodes) and issues an authorization decision which the PEP has to enforce together with some (optional) obligations and/or advices. Given its potentially limited resources, a PEP might not have the capacity to support the context handler functionality, which can be offloaded to a PDP or another infrastructure component, such as a power node.

4.1.3.3 Infrastructure entities (Power node)

On the power nodes, other than the PDP and PIP/PAP implementations, development will also include a bundle for deploying DPWS Devices and also a DPWS Client bundle. The latter will be used to identify DPWS devices in the network and register to any services they may offer. An example of former, namely the creation of an Airconditioner device via the respective OSGi bundle, can be seen in Figure 4-14



Device IP	enter device ip
Device UUID	enter device UUID
Device Label	Device Label

Figure 4-14: Deploying AC_Device on Knopflerfish

4.2 SHIELD policies definition

The purpose of this section is to show how policies are considered inside nSHIELD project in part as input to a Policy-based management which aim to ensure a defined level of security, privacy and dependability in part as a simple set of conditions to serve as the governing reference for any required adaptation a particular scenario may require and how this different point of view with a same target can work on parallel levels to obtain a common result.

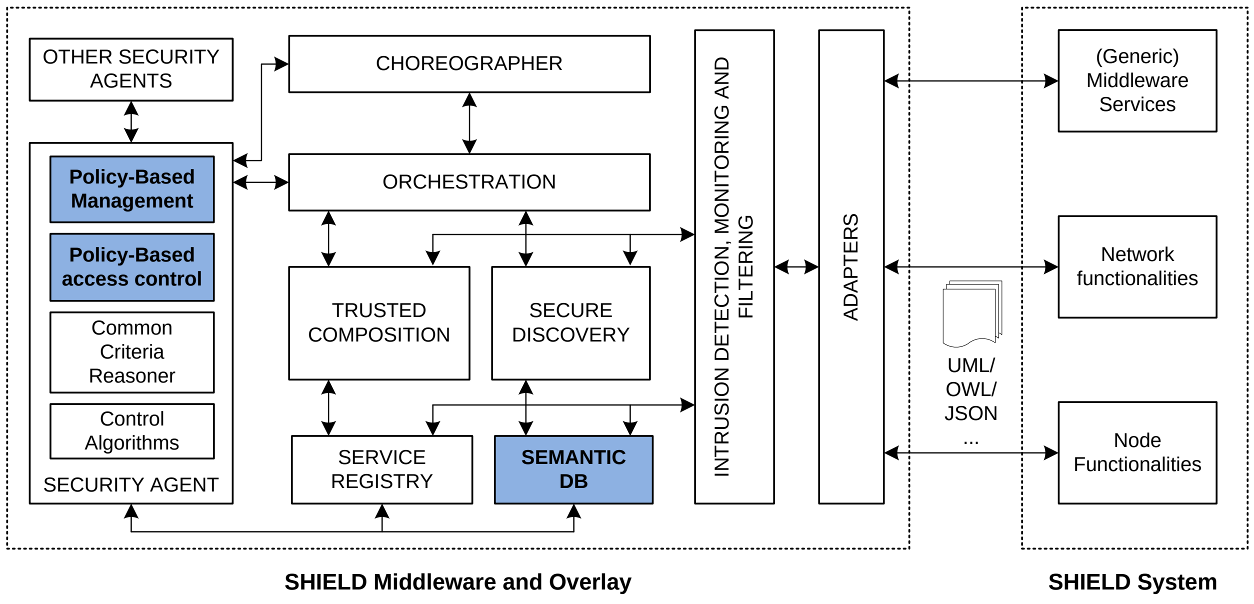


Figure 4-15: SHIELD policy definition

4.2.1 Description

To analyse the concept of policies and its use, it is necessary to first clarify the meaning of the notion of policy and derive criteria for policy classification. At this purpose in the following paragraphs [45] the following concept will be analysed:

- different definitions of policies with evidence of their strengths and weaknesses
- aspects of policy classification (motivation, criteria for classification and hierarchy)

4.2.1.1 Differences in policies definition

In general as a system starts to become more complex (heterogeneity, distribution, etc…) it needs management and two main notions made their way into the literature: domain and policy. They were introduced to reduce the complexity of management tasks but their interpretations vary between and within the fields of research, standardization and industry.

Following the common idea to model network devices, systems, and applications in an object oriented fashion as Managed Objects (MOs), the ISO in its drafts on standardizing domains and policies ([46], [47]) places MOs in a logical domain provided they satisfy a certain policy. Thus a policy is merely a number of rules tied in to a domain managed object.

A very similar approach can be found in the ODP draft standards [48] where domains are just sets of objects that satisfy a specific policy.

In the field of research a different approach is taken. The definition of policy is independent from that of a domain and hence a policy may be either applied to or used to define a domain.

[49], [50] define policy as to influence the behaviour of a manager or managed objects. The definition that will use is that policies are derived from management goals and define the desired behaviour of distributed heterogeneous systems, applications, and networks ([51]). The difference between the above two definitions is simply the level of abstraction.

Applying policies to MOs is already at a technical level, whereas specifying the behaviour of the overall environment is at a higher level of abstraction and more appropriate for enterprise management tasks. Concerning the definition of policy services, [52] specifies some services which could be used in writing management applications using the concept of policies.

To summarize, the advantages of the research-definition are as follows:

- policy and domain are two independent concepts which may but need not be combined;
- policies may be used to define a domain but may also be applied to a domain of objects; and
- policies are an active concept. Policies can initiate or change the characteristics of on-going management activities.

4.2.1.2 Aspects of Policy Classification

4.2.1.2.1 115BMotivation

As interest grows and research in the field of complex systems management progresses, it becomes increasingly important to clarify what means management policies.

A first step towards this was the definition of the term policy. However, this only allows to distinguish whether a statement is considered to be a policy or not ([53]).

The vast number of policies, some examples of which will be presented later, calls for a classification i.e. a well-defined set of (orthogonal) grouping criteria. The goals of such a classification of policies are listed below:

1. to find commonalities of and similarities between different classes of policies;
2. to derive and verify the components of a formal definition of policies;
3. to derive a policy hierarchy for the process of policy refinement and transformation;
4. to allow a (semi-) automatic processing (enforcement and monitoring) of policies; and
5. to guarantee an efficient and effective management of policies.

The manner in which a policy is applied can be very different depending on the class of policy, i.e. its characteristics. For example, the classification aspect *life-time of policy* can be used to distinguish between short, medium, and long-term policies for which the realization may range from a polling strategy to check the enforcement of a short-term policy, to a more complex configuration of asynchronous notifications (traps) in network probes and proxy systems for long-term policies. Thus, our goal is to define classification criteria which influence or even determine the way in which certain classes of policies are defined, activated, enforced, monitored, changed, deactivated, and managed or the way in which policy conflicts are resolved. This then leads to the definition of policy services, in other words services used by management applications to employ the concept of policies. Thus, a precise and detailed classification is a prerequisite in the process of deriving parameters for policy templates, policy processing, and their application.

4.2.1.2.2 Criteria for the Classification of Policies

Extensive analysis of policy catalogues from numerous network and system providers (such as the FidoNet, VirNet, etc.) and talks with network and system managers, administrators and operators (LRZ, BMW, debis) have allowed researcher of University of Munich to collect the following list of criteria for the classification of policies which are illustrated in Figure 1, in form of a multi-dimensional diagram. The precise labels of the axes, i.e. the different categories for each criterion, inevitably depend on the type of target objects the policy deals with and the functional area to which the policy can be assigned. Thus, the criteria are not always perfectly orthogonal. This aspect will be discussed further in Section 4.2.1.2.3.

- **Time:** Time considerations are important because a policy for example may be active throughout the lifecycle of its target objects or may only be activated for a short while, e.g. at the start when a new network device goes into operation (configuration policy). Policy enforcement instructions may need to be dispatched repeatedly, e.g. whenever new devices are added to the configuration. Further refinement of this criterion are:
    - **Life time:** The duration of a policy may be characterized by a short, medium, or long-term application (i.e. enforcement and monitoring). A more detailed consideration of this idea can be based on separating policy enforcement and policy monitoring [51], both of which may be short, medium, or long-term activities. Short-term policy application in this context may be a once-only management policy enforcement (e.g. which user interface to install on a PC), medium-term for example a policy applicable until a new software release is installed, and long-term for example once-and-for-all policies applicable to a compute-server independent from its replacement by a different vendor’s system. As stated above, these categories must be put in concrete terms depending on the type of target object(s) affected by the policy.
    - **Trigger mode:** The question here is whether the enforcement and especially the monitoring are constantly active, repeated periodically for a specific time interval, triggered by asynchronous events or a combination of the last two. Another aspect could be a policy’s relationship to other policies. Examples of this are: no relationship, sequential execution, simultaneous execution, etc.

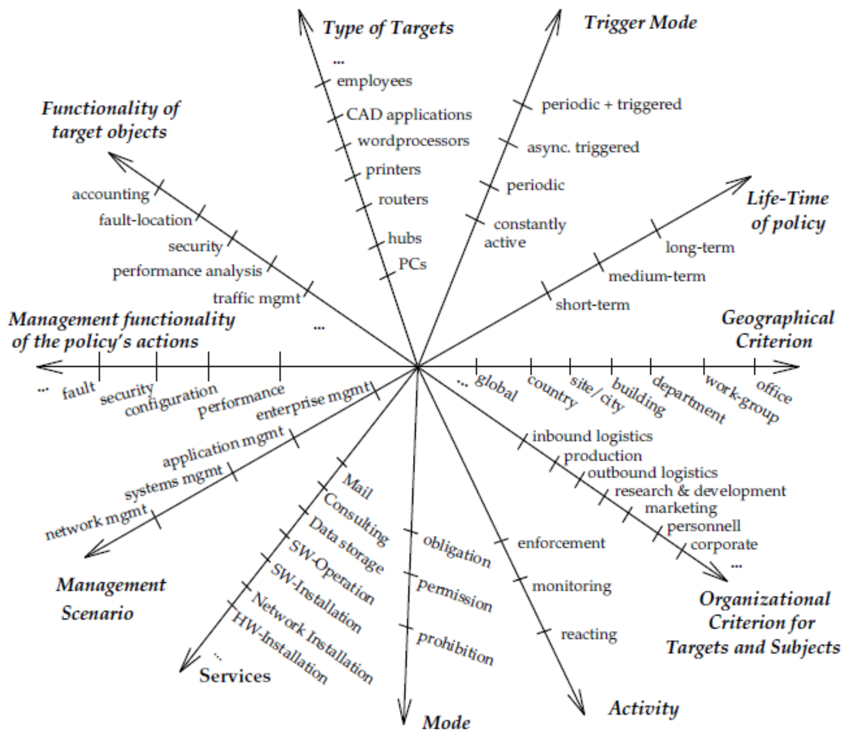


Figure 4-16: Criteria for policy classification

- **Activity:** A policy may be only monitoring or enforcing actions on its target objects or reacting to an event. The monitoring policy will only report its observations but never take on actions whereas enforcing and reacting policies can initiate management activities (actions and reactions).
- **Mode:** Policies can be a constraint or an empowerment. We will use the distinction between an obligation, permission, and a prohibition.
- **Geographical criterion:** With this criterion, policies are grouped according to their location, i.e. affecting co-located physical and logical resources along geographical boundaries. Examples are policies for systems within a LAN segment, all systems in a virtual private network. Typical aspects for a grouping of systems and resources are also: office, work-group, department, building, site/city, country, global.
- **Organizational criterion:** This grouping of policies reflects the organizational structure of the environment, e.g. policies for specific business units of an enterprise or policies which only need to be obeyed in high security departments, or policies in a wider view. Other categories such as inbound logistics, operations, outbound logistics, marketing and sales, service, procurement, research and development, or corporate can be derived from the value chain [54] of an organization. The category ’corporate’ would qualify a policy for the overall corporation and need to be obeyed in high security departments. Other categories such as inbound logistics, operations, outbound logistics, marketing and sales, service, procurement, research and development, or corporate can be derived from the value chain [54] of an organization. The category ’corporate’ would qualify a policy for the overall corporation.
- **Service criterion:** Policies are often specific to certain services an organization offers its customers, buys from service providers, or provides for internal use. Thus, the service for which a policy is specified can help to identify a certain set of management tools to be used or the resources to be taken action on in order to enforce the policy. Services are data storage, email, network information services or software installation to name just a few.
- **Type of targets:** This criterion could include policies applicable to all end systems from one vendor, or all PCs in one department, i.e. target objects with common characteristics. Familiar categories here are resources such as workstations, PCs, hubs, router, laser printers, word processing applications, CAD applications, employees, etc.
- **Functionality of targets:** This includes all policies which apply to resources with a set of common functionalities although possibly of different characteristics otherwise. Targets with common characteristic functionalities could be all network devices capable of routing, all systems (PCs, printers, hubs, etc.) whose user-interface is protected by a password mechanism. In other words, functional characteristics such as accounting, fault-location, security, traffic management, performance analysis etc.
- **Management scenario:** Policies may be associated with a particular management scenario such as network management, systems management, or application management. Some policies may overlap and should thus be grouped together as enterprise management policies. For the scenario of network management, a group of policies may be those applicable to one specific layer of the OSI basic reference model [55] or policies applicable to several adjacent layers. The next criterion is based on the management scenario:
- **Management functionality within a management scenario:** Within each of the above scenarios, we can distinguish different functional areas, e.g. configuration management within systems management, or configuration management within network management, or security management for enterprise management. As mentioned earlier, the above criteria can be used to derive components (attributes) of a policy template. However, the values of these attributes depend on the policy’s level of abstraction i.e. its position in the policy hierarchy.

4.2.1.2.3 Policy Hierarchy

Before presenting a few examples with their classification, the important issue of a policy hierarchy or better the level of abstraction must be raised. When analysing policies, we must differentiate between (see also Figure 4-17 and [56]):

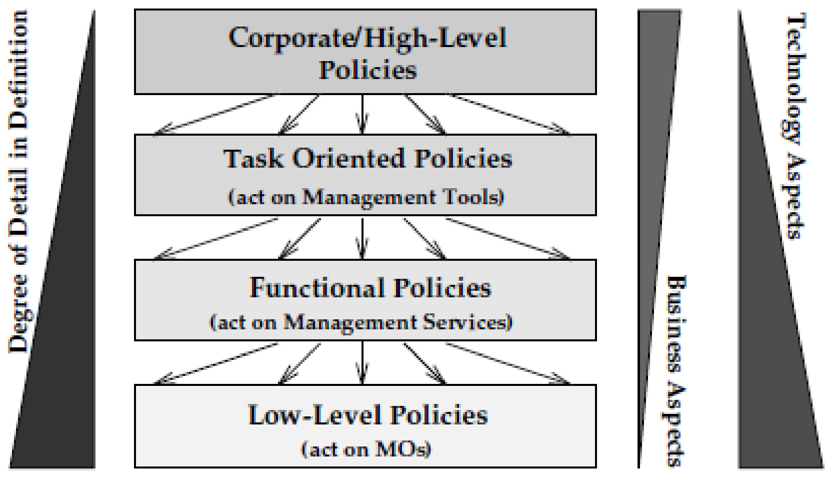

Degree of Detail in Definition
Corporate/High-Level Policies
Task Oriented Policies
(act on Management Tools)
Functional Policies
(act on Management Services)
Low-Level Policies
(act on MOs)
Business Aspects
Technology Aspects

Figure 4-17: The policy hierarchy

- Task oriented policies: Their field of action is sometimes referred to as task or process management, where they define the way how management tools are to be applied and used to achieve the desired behaviour of the resources.
- _Functional policies: These policies operate at the level of and define the usage of management functions, such as the OSI systems management functions ([57]), the OSF/DME distributed services ([58]), or OMG's object services ([59] and [60]); and
- _Low level policies: They operate at the level of managed objects (MOs). MOs in this context refer to simple abstractions of managed network, system resources, and not MOs for e.g. systems management functions.

Thus we find "simple policies" with MOs as their target objects, policies that operate on more complex managed objects such as SMFs, and policies that act on task objects (e.g. abstractions of management tools). We will not consider corporate policies any further in this paper but believe that the other three types of policies can be derived from these. A policy definition (or a policy object) will have the same components at each level in the hierarchy, but the possible values for each component / attribute will depend on the level of abstraction. In other words, the lower the level of abstraction, the more precise and detailed will the definition become, i.e. the granularity of the criteria increases.

The transformation process of a policy definition is a process of stepwise refinement, moving from high-level policy definitions down to low-level (MO-based) policy definitions which can be more easily automated and applied to the managed environment. The question to whether this transformation process can be automated cannot be answered at this stage. However, for any automation of this process (fully computerized or human guided), management information on the managed environment and the management capabilities of the involved systems is necessary.

Some policies can be assigned to exactly one level of the hierarchy, yet other (less well defined) policies can be assigned to different levels of abstraction and thus need to be split into separate policies before the transformation process can be applied.

The application of this kind of approach (each policy can be classified and identified by a hierarchical point of view) can be applied to nSHIELD project as indicated in the following description.

In the most easy condition, for each SPD functionality defined in a particular nSHIELD compliant embedded system, the Overlay decide what is the correct implementation of the particular SPD functionality (considering defined low level policies that characterized that SPD functionality) to obtain a defined SPD level. But this easy solution could not be valid due to external factors which involve in an active or passive way policies that are hierarchically higher. In particular two main situations can occur:

a. The solution, identified by overlay engine, cannot be used due to the presence of a constrain imposed by a higher level policy (passive involvement of a policy – its existence block a defined solution); in this case the overlay engine must define a new solution

b. The overlay engine is not able to identify an available solution; in this case an opportune policy definition can fill this gap (active involvement of a policy – its existence determines a solution).

The following schema summarizes this approach.

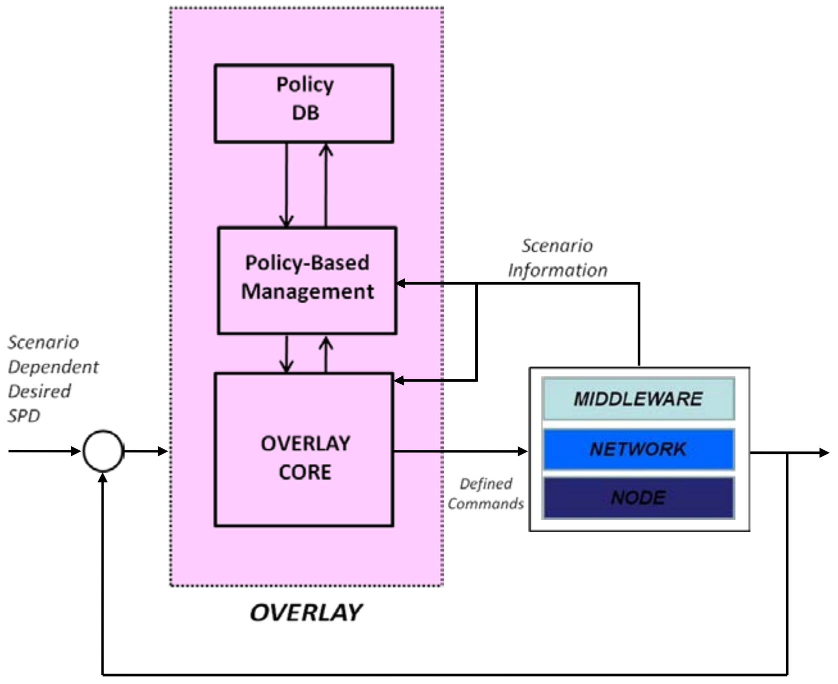


Figure 4-18: nSHIELD compliant System (Scenario dependent) policy involvement

A policy is constituted by a number of rules that are the most elementary unit of a policy, so to implement the solution highlighted in the previous schema, it is necessary to define a methodology to describe a policy and their single rules that an nSHIELD compliant embedded system must satisfy in its particular scenario. These rules, described in a formal way, will constitute the input of the Policy Based Management system. As defined in paragraph 4.1 the solution adopted for secure policy based management is based on eXtensible Access control Markup Language (XACML) policies. XACML is an XML-based general-purpose access control policy language. In the following paragraph will be briefly described how XACML implements a single policy rule.

4.3 XACML policy implementation for PBAC

The main components of the XACML Policy are depicted in Figure 4-19. A policy set about a specific target consists of a number of applicable policies which in turn define a set of rules. Each rule contains information about the applicable Target, the effect, and additional Conditions. Note that the target does not only refer to resources. It might reference characteristics of a subject, resource, action or environment.

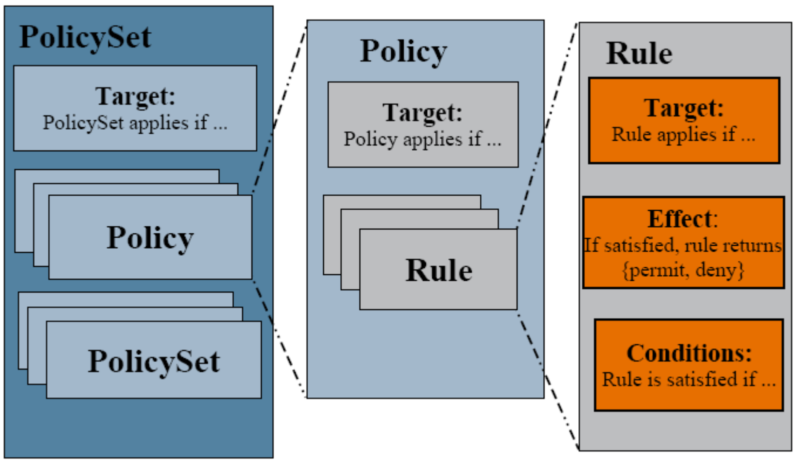


Figure 4-19: XACML Policy components (source: Sun Microsystems)

4.3.1 RULE implementation

A ***rule*** is the most elementary unit of ***policy***, which is typically encapsulated within a policy. This also facilitates the exchange of rules among the stakeholders, i.e. PDP and PAP. The main components of a ***rule*** are:

- a ***target***;
- an ***effect***, indicates the ***rule***-writer's intended consequence of a "True" evaluation for the ***rule***. Two values are allowed: "Permit" and "Deny".
- a ***condition***,
- ***obligation*** expressions, and
- ***advice*** expressions

4.3.2 POLICY implementation

R***ules*** are not exchanged amongst system entities. Therefore, a ***PAP*** combines ***rules*** in a ***policy***. A ***policy*** comprises four main components:

- a ***target***;
- a ***rule***-combining algorithm-identifier;
- a ***set of rules***;
- ***obligation*** expressions and
- ***advice*** expressions

4.3.3 Policy Information Template

Table 4-1 defines the policy information template used to represent policies for controlling access to nSHIELD resources. This template is used to facilitate rules and policies defined by PAP (scenario owners in the context of the nSHIELD) to be imported into the system.

Table 4-1: Policy Attributes Template

POLICY ATTRIBUTE	VALUE
Policy ID	A unique identifier that allows the policy to be referenced within a policy set.
Rule Combining Algorithm	The procedure for combining decisions from multiple rules. Valid values for this attribute are defined below.
Description	A textual description of the purpose of the policy. It typically provides information on most of the attributes found in this template.
Policy Target	The part of a policy that specifies matching criteria for figuring out whether a particular policy is applicable to an incoming service request. Contains three basic "matching" components: **Subjects** (An actor*)*, **Actions** (An operation on a *resource)*, and **Resources** (Data, service or system component)**.** Attributes of the subjects, resource, action, environment and other categories are included in the request sent by the PEP to the PDP.
Effect	The intended consequence of a satisfied rule. It can take the values "Permit" and "Deny". Note that this is not necessarily the authorization decision returned by the PDP to the PEP which, besides the above, can also include the values “Indeterminate” or “NotApplicable", and (optionally) a set of ***obligations and advices***
Condition (optional)	Represents a Boolean expression that refines the applicability of the ***rule***
Obligations expressions (optional)	Operation that should be performed by the PEP in conjunction with the enforcement of an authorization decision.
Advice expressions (optional)	A supplementary piece of information in a policy or policy set which is provided to the PEP with the decision of the PDP.

4.3.4 Rule- and Policy-combining algorithms

The following algorithms are used for combining rules of a policy as well as policies from a policy set.

- **Deny-overrides:** It is intended for those cases where a deny decision should have priority over a permit decision
- **Ordered-deny-overrides:** The behaviour of this algorithm is identical to that of the “Deny-overrides” rule-(policy-) combining algorithm with one exception. The order in which the collection of rules (policies) is evaluated SHALL match the order as listed in the policy (set).

- **Permit-overrides:** It is intended for those cases where a permit decision should have priority over a deny decision.
- **Ordered-permit-overrides**: The behaviour of this algorithm is identical to that of the “Permit-overrides” rule-(policy-) combining algorithm with one exception. The order in which the collection of rules (policies) is evaluated SHALL match the order as listed in the policy (set).
- **Deny-unless-permit:** It is intended for those cases where a permit decision should have priority over a deny decision, and an “Indeterminate” or “NotApplicable” must never be the result. It is particularly useful at the top level in a policy structure to ensure that a PDP will always return a definite “Permit” or “Deny” result. This algorithm has the following behaviour.
    1. If any decision is "Permit", the result is "Permit".
    2. Otherwise, the result is "Deny".
- **Permit-unless-deny:** It is intended for those cases where a deny decision should have priority over a permit decision, and an “Indeterminate” or “NotApplicable” must never be the result. It is particularly useful at the top level in a policy structure to ensure that a PDP will always return a definite “Permit” or “Deny” result. This algorithm has the following behaviour.
    - 1. If any decision is "Deny", the result is "Deny".
    - 2. Otherwise, the result is "Permit".
- **First-applicable (rule):** Each rule SHALL be evaluated in the order in which it is listed in the policy. For a particular rule, if the target matches and the condition evaluates to "True", then the evaluation of the policy SHALL halt and the corresponding effect of the rule SHALL be the result of the evaluation of the policy (i.e. "Permit" or "Deny"). For a particular rule selected in the evaluation, if the target evaluates to "False" or the condition evaluates to "False", then the next rule in the order SHALL be evaluated. If no further rule in the order exists, then the policy SHALL evaluate to "NotApplicable".
- **First-applicable (policy):** Each ***policy*** is evaluated in the order that it appears in the ***policy set***. For a particular ***policy***, if the ***target*** evaluates to "True" and the ***policy*** evaluates to a determinate value of "Permit" or "Deny", then the evaluation SHALL halt and the ***policy set*** SHALL evaluate to the ***effect*** value of that ***policy***. For a particular ***policy***, if the ***target*** evaluate to "False", or the ***policy*** evaluates to "NotApplicable", then the next ***policy*** in the order SHALL be evaluated. If no further ***policy*** exists in the order, then the ***policy set*** SHALL evaluate to "NotApplicable".
- **Only-one-applicable (for policies only):** In the entire set of *policies* in the *policy set*, if no *policy* is considered applicable by virtue of its *target*, then the result of the policy-combination algorithm SHALL be "NotApplicable". If more than one *policy* is considered applicable by virtue of its *target*, then the result of the policy-combination algorithm SHALL be "Indeterminate". If only one *policy* is considered applicable by evaluation of its *target*, then the result of the *policy-combining algorithm* SHALL be the result of evaluating the *policy*.

4.3.5 Policy examples

4.3.5.1 Policy classification and identification by a hierarchical point of view

The following examples are taken from real life situations from different corporations where the operators applied these policies without a systematic and structured approach. Thus, the values for each classification criterion were derived manually, since none of these policies were systematically refined. For each example, the level of abstraction is given and possible values for each of the above classification criteria are indicated. Examples 1 and 2 are used to show the components of a policy definition, whereas example 3 illustrates the splitting of a “composite” policy into separate policies after which the transformation and refinement process can be applied.

Example 1:

”The exchange of data between the company’s headquarters and its remote production sites is to be done between 18:00 and 22:00 hours in encrypted mode.” The degree of detail in this policy is very limited and thus, we can only record it as a high level policy of the following format with several dimensions to be further specified:

- **level of abstraction**: high level policy
- **classification criteria**:
    - *life time:* long-term (no end specified)
    - *trigger mode:* periodic (daily between 18:00 and 22:00 hours)
    - *activity:* enforcement (no reaction is specified if the time interval or the security level are not *obeyed – a separate policy for this purpose would be necessary)*
    - *mode:* obligation
    - *geographical criterion:* corporate headquarters and production sites
    - *organizational criterion: unspecified*
    - *service criterion: unspecified*
    - *type of targets: unspecified*
    - *functionality of targets: unspecified*
    - *management scenario: enterprise management*
    - *management functionality within a management scenario: security management for enterprise management*

Analysing and refining this policy further leads to a number of low level policies, depending on the way the encryption is achieved. The following two policy descriptions illustrate this, the first enforcing the encryption by activating either encryption modems or scramblers, the second by activating the encryption mode for data transfer in the application software. This also shows that a policy can be applied in several different ways without changing the management goal.

- **level of abstraction**: low level policy

  This is because the policy applies to MOs which, in this case, are abstractions of network devices, i.e. modems or scramblers
- **classification criteria:**
    - *life time:* long-term (no end specified)
    - *trigger mode:* periodic (daily between 18:00 and 22:00 hours)
    - *activity:* enforcement
    - *mode:* obligation
    - *geographical criterion:* corporate headquarters and production sites
    - *organizational criterion:* networking department
    - *service criterion:* data transfer service
    - *type of targets:* **encrypting modems or scramblers**
    - *functionality of targets:* data transfer or encryption
    - *management scenario:* network management
    - *management functionality within a management scenario:* security management for network management
- **level of abstraction**: low level policy

This is because the policy applies to MOs which, in this case, are abstractions of the application software based on a client-server architecture e.g. distributed CAD or word processing applications.

- **classification criteria**:
  - *life time:* long-term (no end specified)
  - *trigger mode:* periodic (daily between 18:00 and 22:00 hours)
  - *activity:* enforcement
  - *mode:* obligation
  - *geographical criterion:* corporate headquarters and production sites
  - *organizational criterion:* systems department
  - *service criterion:* application software installation and software maintenance
  - *type of targets:* general distributed applications based on a client-server architecture, which therefor transfer data across the network.
  - *functionality of targets:* **applications with encryption**
  - *management scenario:* application management
  - *management functionality within a management scenario:* security management for systems and application management

Looking back at the policy hierarchy introduced in Section 4.2.1.2.3, it can be noted that the above policy was refined to neither different low-level MO-based policies, without specifying task oriented policies nor functional policies. This is because there were no management tools or management functions which could have been used to enforce this policy at a higher level. However, if these had been available, a task oriented policy could have specified the way to use a management tool for the configuration of modems or scramblers, or a functional policy could have defined the manner in which to use a certain encryption management function.

<u>Example 2:</u>

”If workstation access is protected by a password mechanism, passwords must be at least 6 characters long, if they combine upper-case and lower-case letters, or at least 8 characters long, if in monocase. No other password structure is allowed.”

- **level of abstraction:** managed-object based policy

  This is a low-level or managed-object based policy, as it specifies the characteristics of the specific password mechanism, i.e. a specific implementation of e.g. an authentication management function. Provided a Managed Object for the password mechanism exists, the policy can already be used to set the attributes’ values. It is not a functional policy, because the attributes and not the functionality of the password mechanism are affected by the policy.
- **classification criteria**:
  - *life time: long-term*
  - *trigger mode: asynchronously triggered (e.g. by execution of the UNIX command passwd)*
  - *activity: monitoring, reacting (to a wrong password structure), and enforcing (setting the password mechanism’s characteristics)*
  - *mode: obligation*
  - *geographical criterion: global*
  - *organizational criterion: corporate*

- *service criterion: data processing (authentication)*
- *type of targets: workstations*
- *functionality of targets: authentication/password mechanisms*
- *management scenario: systems management*
- *management functionality within a management scenario: security management within systems management*

Example 3:

”Travel agencies are to be connected to the central booking office through leased lines. In case of failure, dial-in lines are to be provided, and the agencies must authenticate themselves with their login-IDs and login-keys.”

This policy obviously mixes aspects of several levels of abstraction, the level of corporate policies, the level of functional policies, and the level of MO-based policies. The policy should be split into separate policies of specific levels of abstraction e.g.: (3a, corporate) ”the network operations center at the central booking office is to provide and maintain leased lines to the agencies, and modems for dial-in connections”, (3b, functional) ”in case of failure of a leased line, modems are to be activated for dial-in connections”, (3c, functional) ”dial-in connections are to be protected by an authentication procedure.” and (3d, MO-based) ”the authentication mechanism MO must guarantee the use of non-empty login-ids and login-keys”. For the sake of brevity we will not discuss the classification of these policies here further. Yet, these examples clearly show that this classification allows us to find commonalities among policies and that this form of classification is a necessary first step towards finding the components of a formal policy definition. The transformation process will only be able to refine some components/attributes further, depending on the management information available to the process.

4.3.5.2 XACML Policy implementation example

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
    xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
        access_control-xacml-2.0-policy-schema-os.xsd"
    PolicyId="urn:oasis:names:tc:xacml:2.0:conformance-test:IIA1:policy"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
overrides">
  <Target/>
  <Rule
      RuleId="urn:oasis:names:tc:xacml:2.0:conformance-test:IIA1:rule"
      Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch
              MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#string">nshield_user</AttributeValue>
            <SubjectAttributeDesignator
                AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
```

```
        <Resource>
          <ResourceMatch
              MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
            <AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#anyURI">Freight_ACDevice</AttributeValue>
            <ResourceAttributeDesignator
                AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch
              MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#string">SetTemperature</AttributeValue>
            <ActionAttributeDesignator
                AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
  </Rule>
</Policy>
```

5 SHIELD Overlay

5.1 Proposed SHIELD Security Agents architecture

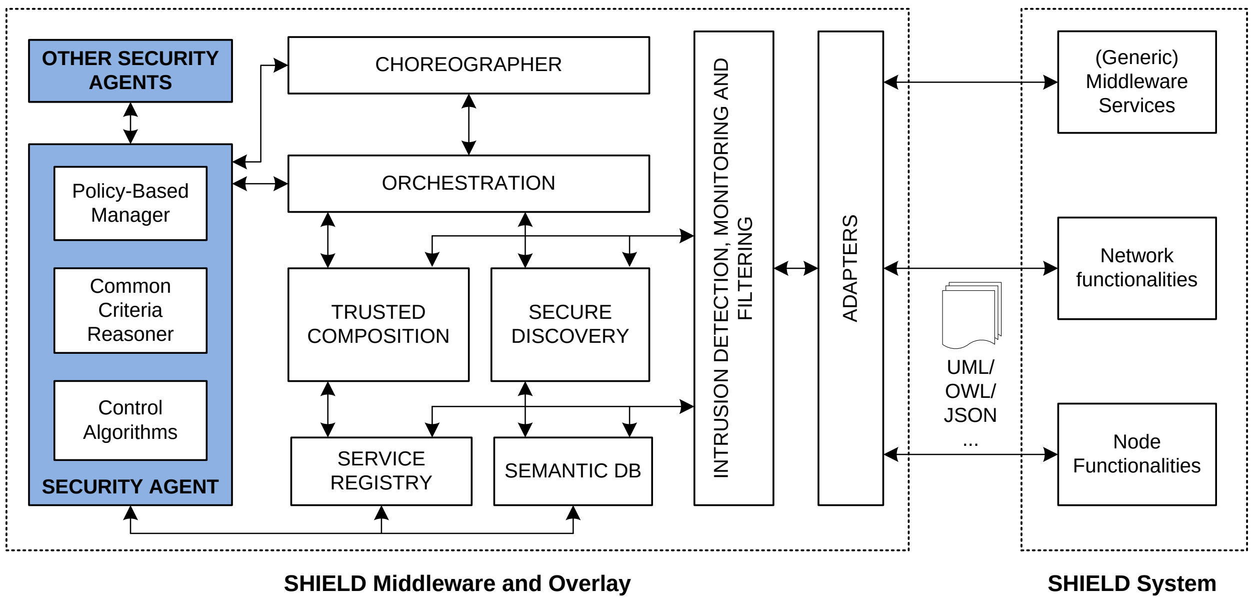


Figure 5-1: SHIELD Security Agent

An overlay agent has been developed and deployed on power nodes. The agent's main functionality will be to monitor and log the status of various services running on the power node itself and other (i.e. micro & nano) nodes under its supervision.

Critical services running on nSHIELD nodes will be monitored by individual daemons. The daemons will be responsible for restarting said services - when necessary - and will also inform the overlay agent of their services' status, restarts etc. The overlay agent will log these daemon messages and facilitate their assessment via a reporting mechanism (e.g. a GUI).

Development of the overlay agent will include a proof of concept implementation using one of nSHIELD's available services (e.g. anonymity).

5.1.1 Description

In security environment, the composability problem, as defined by McLean in [61], is widely investigated:

> *"A general ability to build composite high-assurance systems presupposes a general theory of system composition. Such a theory provides insight into why certain properties are preserved or not preserved by certain forms of composition. More importantly, for a large class of properties and a variety of composition constructs, it answers questions of the form: "if a system satisfying property X is composed with a system satisfying property Y using composition construct Z, what properties will the composite system satisfy?"* [61]

In other words, in the literature the composability problem is generally used to refer to the study of how the composition affect the security properties, (in particular information flow properties) of single component, that composes a complex system, in general the relation between complex system properties and single component properties. In [62] the problem of composability of the non-interference properties is discussed, the author shows that by connecting two systems, both of which are judged to be secure, it is possible that the composite system is not secure. In [63] and [64] a general theory on security properties is presented, in particular in [64] the authors compare their framework with McLean's Selective Interleaving Functions framework [61]. Finally, in [65] various compositionality results are presented, in details the author demonstrates that certain nontrivial security properties emerge under composition and illustrates how this fact can be exploited.

In the SHIELD project we take into account the composition to assure the desired SPD level, then the focus is on the composability of SPD functionality.

In the literature a wide array of Security and Dependability models and model-based evaluation are available (e.g. [66], [67], [68], [69]). In [70] a methodology to construct dependability models, using generalized stochastic Petri nets ([71], [72], [73]) and stochastic reward nets, is described. This topic is investigated by Jonsson (e.g. [69], [74]), in particular, in [69], he proposes a conceptual model to define security and dependability characteristics in terms of system's interaction with its environment. In [67] the authors survey the model-based techniques for the system dependability evaluation and summarize how to extend them to security domain. This paper shows that many dependability evaluation techniques can be used to evaluate system security, but some problems have not yet been fully overcome (due to fundamental differences between the accidental nature of the faults commonly assumed in dependability evaluation, and the intentional, human nature of cyber-attacks). Finally, in [66] the authors aim to meet the need to quantify system properties methodically, in fact they present a threats classification and various types of individual models that can be combined and represented by one of the common modelling method (e.g. Markov chains, stochastic Petri nets).

In the pilot phase, the composition problem was successfully modelled by means of Hybrid Automata Theory and solved by two different approaches: a static approach with simple optimization and an operating conditions approach with MPC control. The main limits of the hybrid automata approach are: the scalability problem, the implementation issue and the limitations to represent the complexity of embedded systems (abstraction is needed).

To overcome these problems we investigate Discrete Event System (DES) problem formalization.

A Discrete Event System (DES) is a discrete-state event-driven system, in which the state evolution is determined by asynchronous events that occur over the time. In particular, a DES is characterized by: (i) the feasible events set E; (ii) the state space constituted by the discrete set X and (iii) the state transition mechanism (event-driven).

Formally DES can be modelled by Automata and Petri Nets (PN). The former is the basic DES model ([75], [76]), that, on the one hand, is characterized by an intuitive structure, easy composition operations, and ease of analysis, but, on the other hand, it suffers from state space explosion (automata model should be not adequate for complex system). The latter modelling formalism PN was introduced by C. A. Petri in [77], it allows (i) the representation of a larger class of systems (automata can always be represented as PN, but the contrary is not true), (ii) to explicit the conditions to enable the event and (iii) to overcome the state space explosion. In particular an extension of PN, the Coloured Petri Nets (CPN) ([78], [79], [80]), are developed to model complex systems, in fact, the CPN combine the PN structure with the high-level programming, i.e., using data types and complex data manipulation. Furthermore CPNs allows obtaining hierarchical descriptions of system, combining a set of sub-models and defining the interfaces between these sub-models.

In the following box, an introduction of formal description of DES is provided; this text is fully taken from nSHIELD Deliverable *D5.1: SPD middleware and overlay technologies assessment*.

Automata Formal Description

Considering DES formalism ([75], [76]), 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*. In other word, a language, defined over an events 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*.

Definition 1: A *Deterministic Automaton G* is a five-tuple $G = (X, E, f, x_0, X_m)$

where:

- X is the set of states;
- E is the finite set of events associated with G
- $f : X \times E \rightarrow X$ is the transition function: $f(x, e) = y$ means that there is a transition, labelled e, from state x to state y;
- x_0 is the initial state;
- $X_m \subseteq X$ is the set of marked states (final states).

A deterministic Automaton is defined in few works (i.e., [75]) as a six-tuple $G = (X, E, f, \mathrm{T}, x_0, X_m)$, where T is the feasible event function defined as T: $X \rightarrow 2^E$, $\mathrm{T}(x)$ 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.

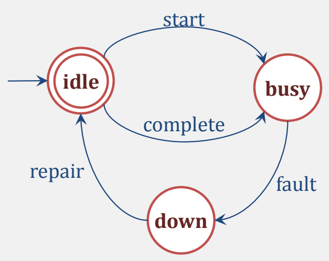


Figure 5-2: State transition diagram of queuing system with breakdown

For example, the simple automaton in Figure 5-2 represents an elementary queuing (or machine) system with breakdown. The system is defined by the following model:

X = {idle, busy, down};	f (idle, *start*) = busy;
E = {*start*, *complete*, *fault*, *repair*};	f (busy, *complete*) = idle;
x_0 = idle;	f (busy, *fault*) = down;
X_m = {idle};	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 or 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 $G_1 \| G_2 := (X_1 \times X_2, E_1 \cup E_2, f, (x_{01}, x_{02}), X_{m1} \times X_{m2})$

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

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.

Petri Nets Formal Description

Like an automaton, a Petri net [81] is a formalism to describe Discrete Event System behaviour; in particular they represent the DES transition function.

Definition 2: A *Petri net* is a five-tuple (P, T, A, w, M_0)

where

- $P = \{p_1, p_2, \ldots, p_n\}$ is the finite set of places;
- $T = \{t_1, t_2, \ldots, t_m\}$ is the finite set of transitions, such that $P \cup T = \emptyset$ $P \cap T = \emptyset$;
- $A \subseteq (P \times T) \cup (T \times P)$ 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 $O(t_i) = p_j$);
- $w : A \rightarrow \{1,2,3,\ldots\}$ is the weight function on the arcs;
- $M_0 : P \rightarrow \{0,1,2,3,\ldots\}$ 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 $w(p_i, t_i)$ tokens from each input place p_i ($p_i = I(t_i)$) and put $w(t_i, p_h)$ tokens in each output place p_h ($p_h = O(t_i)$).

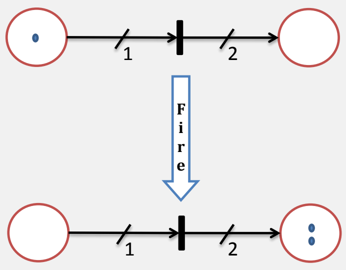


Figure 5-3: n example of enabled transition

Petri Nets allow modelling the typical features of dynamic systems, such as, concurrency, sequential behaviour, synchronization, mutual exclusive, and so on, in Figure 5-4 some examples are shown.

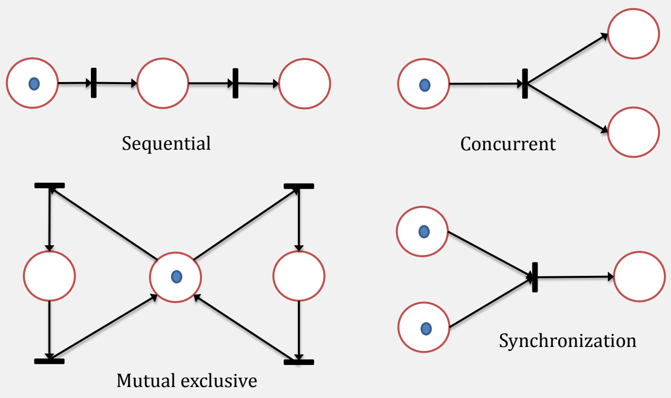


Figure 5-4: Examples of Petri Net primitives

The strength of Petri nets is that they allow to analyse 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* ([82]): 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.

Coloured Petri Nets

Kurt Jensen introduced the Coloured 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 modeled system. The definition of non-hierarchical Coloured Petri Net follows:

Definition 3: A *non-hierarchical Coloured 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 colour sets;
- V is a finite set of typed variables such that $\text{Type}(v) \in \Sigma$ for all variables $v \in V$;
- $C : P \rightarrow \Sigma$ is a colour set function that assigns a colour 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∅ 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 Coloured Petri Net Module follows:

Definition 4: A *Coloured Petri Net Module* is a four-tuple CPNM = (CPN, T_{sub}, P_{port}, P_{type}),

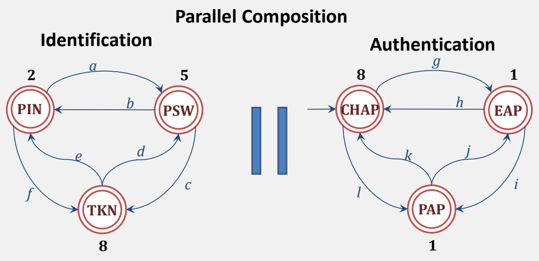
where:

- $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ is a non-hierarchical Coloured 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 [78], [79], [80] for detailed description of the concepts, analysis methods and practical use of coloured Petri nets.

As shown in nSHIELD Deliverable *D5.1: SPD middleware and overlay technologies assessment*, the automata model of SPD functionality suffer scalability problem.

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



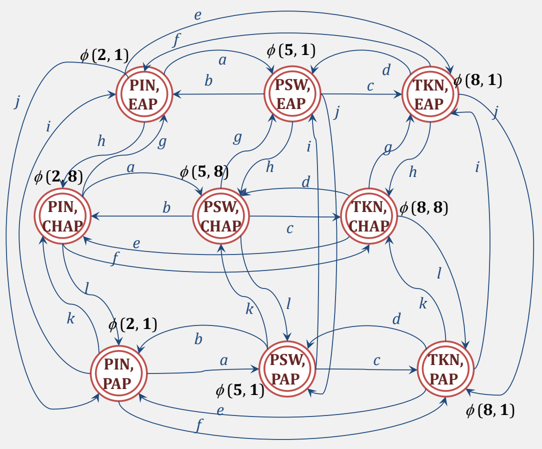


Figure 5-5: SPD functionalities parallel composition

To model an heterogeneous and complex system is necessary: (i) a scalable structure to overcome state explosion and (ii) a simple composition rules to model the SPD composability. In particular, the following table show the guide line to develop nSHIELD overlay composability:

Table 5-1: overlay composability guidelines

Feature	***Description***
SPD Driven	the focus is on SPD components
Scalable	An abstraction of the SPD functionality is necessary to obtain a modular structure
Close to existing standards	Common Criteria
Measurable	Quantification of SPD level
Flexibility	Decision Making at different levels
Deterministic	Formal modelling for composability

CPNs combine the PNs structure with the high-level programming capabilities, furthermore the hierarchical structure is the strength of CPNs, in fact the sub-modules composition allows to model complex system and to work at different abstraction level.

Petri Nets (PN) and Coloured Petri Nets (CPN) are frequently used to model security aspects of system (see e.g. [83], [84], [85]). In [85] the CPN are used to develop a stochastic evaluation system that aims to estimate the integrity of a security critical system. In particular, the CPN formalism is widely used in the industrial and in the information security environment. In [86] is presented examples of the former case. In the latter case, the CPN are mainly employed to the modelling, the verification and the analysis of security protocol (see e.g. [87], [88], [89]) and information system (e.g. [90], [91]). In [90] the authors, using CPN's semantics, propose a formal security model called Secure Coloured Petri.

Considering the guideline in Table 5-1, we propose a basic module (basic element of system) representing the SPD functionality (Figure 5-6). Especially the input state represents the desired value i.e. desired metrics and/or a specific type of implementation.

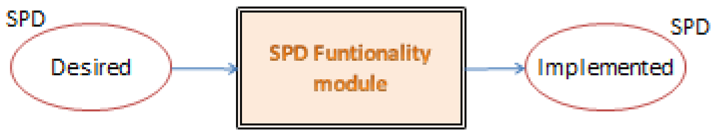
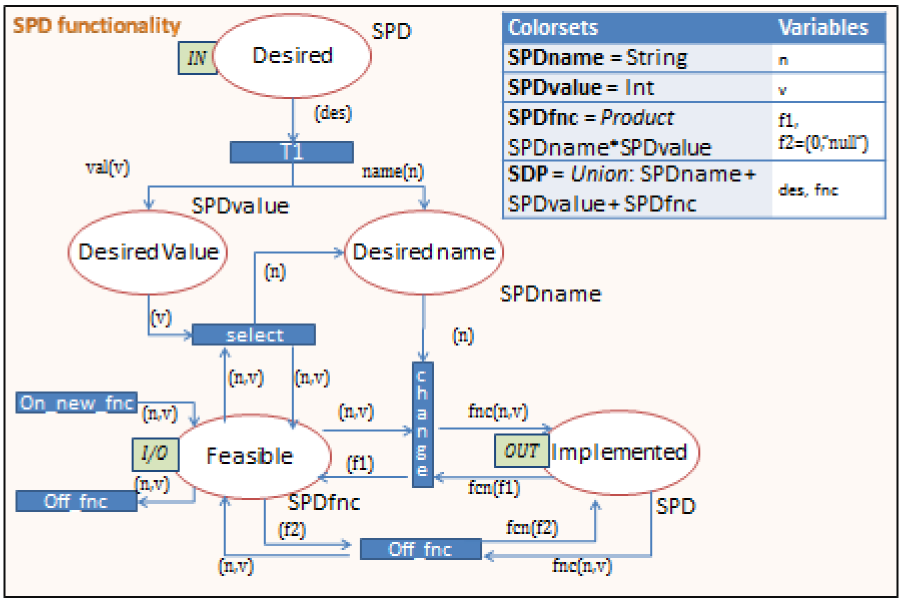


Figure 5-6: the basic module

The Figure 5-6 represents a hierarchical Colour Petri Net, specifically the high level abstraction module. In particular the double-lines border box corresponds to a substitution transition that represents the sub-module (with more detailed view). The input (output) places of substitution transitions are called input (output) sockets. The socket places of a substitution transition are related to the interface of sub-module by means of a port–socket relation. The sub-module that represents the SDP Functionality module is shown in Figure 5-7. This CPN is characterized by places, transitions, arcs, input/output ports, declarations, etc. The place is depicted as a circle characterized by its colour set and optionally by the initial marking and place name. The rectangular boxes represents the transitions while the place-transition and transition-place arcs are depicted as an arrows and associated with the its inscription that defines the tokens and data value affect by transition, in particular how the states change when transition occurs. As previously defined, the CPN allows to work with different abstraction levels, the low level module (Figure 5-7) exchanges tokens with its environment by means input/output port (specific places that constitute the module interface). Port places are labelled with a rectangular port tags specifying the port type: input, output, or input/output port.



Colorsets	Variables
SPDname = String	n
SPDvalue = Int	v
SPDfnc = *Product* SPDname*SPDvalue	f1, f2=(0,"null")
SDP = *Union*: SPDname+ SPDvalue+SPDfnc	des, fnc

Figure 5-7: the sub-page of SPD Functionality module

The Table in Figure 5-7 defines the colour sets

The SPD Functionality module could be simple compose with others as show in Figure 5-8, in which three SPD functionalities are considered.

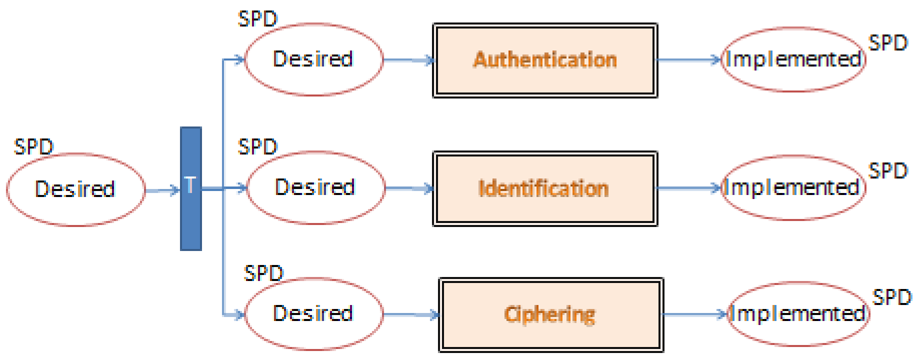


Figure 5-8: a simple composition of SDP modules

In order to complete this model, it is necessary to take into account the possible relation constraint between the implemented functionalities. For example, a particular implementation of the authentication functionality can be used, or simply is effective, only if a precise implementation of identification is enable. We denote this type of relation constraint as "coupling relation". As shown in Figure 4 this relation constraint is modelled as a substitution transition…

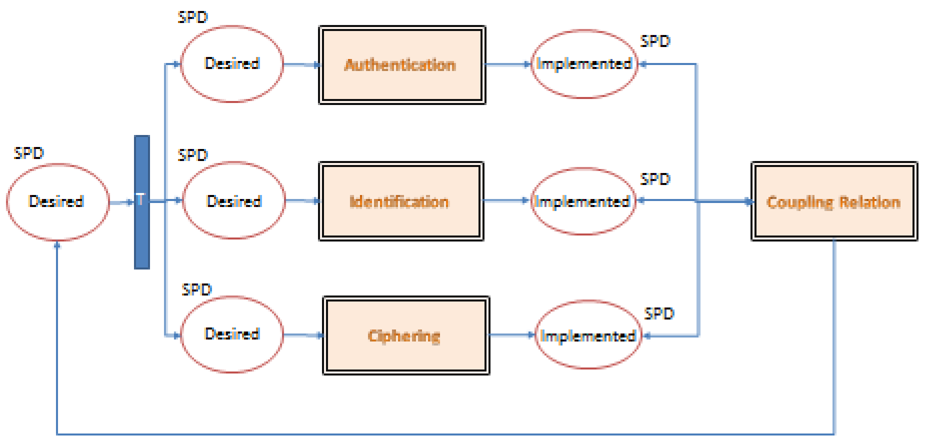


Figure 5-9: nSHIELD model with coupling relation

The Coupling Relation block contains the list of possible coupling constraints on functionality. As shown in Figure 4 this block takes the implemented functionalities as input, verifies the presence of coupling constraint and produces a control action as output. In particular, the control action aims to address any coupling constraint by positioning a new token that carries the name of the implementation to be enabled in order to satisfy the constraint, in the place *Desired.* This token drives the system to satisfy the constraint if the new desired implementation (carried by token) is available and satisfies the required SPD level. If this condition is not verified then the coupling block "turns off" the implementation that required the coupling. Thus the system actives a new implementation to achieve the desired level of SPD, obviously if at least one is available.

6 Conclusions

In this document the preliminary prototypes of the first phase of the nSHIELD project have been presented, reporting the studies carried out, the first design solution and theoretical advances.

These main technologies are:

- A new procedure to derive the SHIELD ontology, based on decoupling between abstraction and domain
- The secure discovery protocol
- The intrusion detection monitoring and filtering service
- The adapters for legacy systems
- The SHIELD Security Agent
- The policy based management architecture
- The policy definition
- The DES and Petri Nets theory for composability
- The protection profile of SHIELD middleware

Some of them have led to the development of real prototypes (i.e. software code, simulations, OWLs, ECC) and these prototypes are collected in D5.2 according to the table included in the next section.

The advances will be addressed in D5.5.

7 Prototypes Table

This tables is used to link D5.2 sections with the associated prototypes

Table 7-1: Prototype tables

Partner	Section of D5.2	Prototype	Type
UNIROMA1	2.1	OWL/ER Diagrams of the SHIELD semantic model	OWL file / Diagrams
MGEP	2.2	Ontology for Intrusion Detection System	OWL file
UNIROMA1	3.1	Protocol for Secure Discovery	Java Code of the OSGI Bundle
SLAB	3.2	Intrusion Detection Bundle	Java Code of the OSGI Bundle
ATHENA	3.3	Adaptation of Legacy Systems	Java Code of the OSGI Bundle
SES	3.4	Middleware protection profile (preliminary)	PP Document
TUC	4.1	Policy Based Access Control Module implementation (preliminary)	Java Code of the OSGI bundle
TUC	4.2	Policy Definition Example	Policy code
UNIROMA1	5.1	Security Agent Implementation (preliminary)	Java Code of the OSGI Bundle
UNIROMA1	5.2	Protection profile	
UNIROMA1	5.2 (Composition Algorithms)	CPN Tool Simulations	Simulations and source code

8 References

[1] R. Anderson, “Why information security is hard - an economic perspective,” in *Seventeenth Annual Computer Security Applications Conference, pp. 358–365*, 2001.

[2] Common Criteria for Information Technology Security Evaluation, *"Part 1: Introduction and general model",* Version 3.1, revision 4, September 2012.

[3] Common Criteria for Information Technology Security Evaluation, *Part 2: Functional security components,* Version 3.1, revision 4, September 2012.

[4] Common Criteria for Information Technology Security Evaluation, *Part 3: Assurance security components,* Version 3.1, revision 4, September 2012.

[5] nSHIELD project team, “Tech. Rep. Deliverable 5.1: "SPD Middleware and Overlay technology Assessment",” European Union FP7 ARTEMIS nSHIELD project, 2012.

[6] S. Morimoto, S. Shigematsu, Y. Goto and J. Cheng, “Formal verification of security specifications with common criteria,” *Proceedings of the 2007 ACM symposium on Applied computing - SAC ’07,* p. 1506, 2007.

[7] D. Herrmann, “Using the Common Criteria for IT Security Evaluation,” in *Auerbach*, 202.

[8] R. Bruni, I. Lanese and U. Montanari, “A basic algebra of stateless connectors,” *Theoretical Computer Science,* no. 366, pp. 98-120, 2006.

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

[10] C. Hoare, “CSP - Communicating Sequential Processes,” in *International Series in Computer Science*, Prentice-Hall, Englewood Cliffs, 1985.

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

[12] S. Bliudze and J. Sifakis, “Causal semantics for the algebra of connectors,” *Formal Methods in System Design, Springer Science,* no. 36, pp. 167-194, 2010.

[13] CONNECT project team, “Deliverable 2.2: "Compositional Algebra of CONNECTors",” European Union FP7 ICT CONNECT Project (http://www.connect-forever.eu).

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

[15] J. Undercoffer, J. Pinkston, A. Joshi and T. Finin, “A Target-Centric ontology for intrusion detection,” in *IJCAI Workshop on Ontologies and Distributed Systems, IJCAI'03*, August, 2003.

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

[17] F. Abdoli and M. Kahani, “Ontology-based Distributed Intrusion Detection System,” in *Development,*

65–70, 2009.

[18] G. A. Isaza, A. G. Castillo and N. D. Duque, “An Intrusion Detection and Prevention Model Based on Intelligent Multi-Agent Systems,” in *Signatures and Reaction Rules Ontologies, pp 237–245*.

[19] G. Isaza, A. Castillo, M. López, L. Castillo and M. López, “Intrusion Correlation Using Ontologies and Multi-agent Systems,” *Information Security and Assurance, Springer Berlin Heidelberg,* no. 76, pp. 51-63, 2010.

[20] O. Corcho, M. López, A. Gómez-Pérez and A. López-Cima, “Building Legal Ontologies with METHONTOLOGY and WebODE,” *Benjamins, V.R., Casanovas, P., Breuker, J., Gangemi, A. (eds.) Law and the Semantic Web. LNCS (LNAI) Springer, Heidelberg,* no. 3369, pp. 142-157, 2005.

[21] A. N. Mian, R. Baldoni and R. Beraldi, “A survey of service discovery protocols in multihop mobile ad hoc networks,” in *Pervasive Computing, IEEE pp.66-74*, 2009.

[22] E. Meshkova and e. al, “A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks,” in *Computer Networks, pp. 2097-2128*, 2008.

[23] E. Guttman, J. Veizades, C. Perkins and M. Day, “Service Location Protocol, version 2,” IETF RFC 2608, June 1999.

[24] E. Guttman, C. Perkins and J. Kempf, “RFC 2609: Service Templates and Service: Schemes.,” Network Working Group, The Internet Society, 1999.

[25] J. Rosenberg and e. al, “SIP: session initiation protocol,” *RFC 3261, Internet Engineering Task Force,* vol. 23, 2002.

[26] National Institute of Standards and Technology (NIST), “186-2. Digital Signature Standard (DSS).,” FIPS, PUB, 2000.

[27] D. Eastlake and P. Jones, “US secure hash algorithm 1 (SHA1),” 2001.

[28] A. A, “Virtual Point-to-Point(TUN) and Ethernet(TAP) devices -,” [Online]. Available: http://vtun.sourceforge.net/tun/.

[29] M. Krasnyansky and M. Yevmenkin, “Virtual Point-to-Point(TUN) and Ethernet(TAP) devices,” Copyright (C) 1999-2008 Maxim Krasnyansky, [Online]. Available: http://vtun.sourceforge.net/tun/. [Accessed 2013].

[30] openslp.org, “What Is Service Location Protocol?,” [Online]. Available: http://www.openslp.org/doc/html/IntroductionToSLP/index.html. [Accessed 2013].

[31] openslp.org, “OpenSLP Programmer's Guide,” [Online]. Available: http://www.openslp.org/doc/html/ProgrammersGuide/index.html.

[32] Oracle Solaris, “Managing Service Location Protocol Services in Oracle Solaris 11.1,” October 2012. [Online]. Available: http://docs.oracle.com/cd/E26502_01/html/E28998/legacy-11.html. [Accessed 2013].

[33] Network Working Group , “Service Location Protocol, Version 2 -- Optional Features (Authentication Blocks),” Vinca Corporation, June 1999. [Online]. Available: http://www.openslp.org/doc/rfc/rfc2608.txt. [Accessed 2013].

[34] openslp.org, “OpenSLP - Service Location Protocol,” [Online]. Available: http://www.openslp.org/.

[35] J. S. Rellermeyer and M. A. Kuppe, “jSLP - Java SLP (service Location Protocol),” [Online]. Available: http://jslp.sourceforge.net/jSLP/index.html. [Accessed 2013].

[36] P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum and A. Wolf, “An Architecture-Based Approach to Self-Adaptive Software,” *IEEE Intelligent Systems,* vol. 3, no. 14, pp. 54-62, 1999.

[37] J. S. Rellermeyer and M. A. Kuppe, “SLP service discovery on OSGi platforms,” [Online]. Available: http://jslp.sourceforge.net/jSLP-OSGi/index.html. [Accessed 2013].

[38] Network Working Group, “RFC2614 - An API for Service Location -,” June 1999. [Online]. Available: http://www.openslp.org/doc/rfc/rfc2614.txt. [Accessed 2013].

[39] Caldera Systems, Incorporated, “The Static Registration File,” open SLP - Service Location Protocol, [Online]. Available: http://www.openslp.org/doc/html/UsersGuide/SlpReg.html.

[40] ZOLERTIA, “Z1 Platform - low-power wireless modules,” [Online]. Available: http://www.zolertia.com/ti. [Accessed 2013].

[41] CrossBow, “IRIS Wirelles measurement System,” [Online]. Available: http://www.dinesgroup.org/projects/images/pdf_files/iris_datasheet.pdf. [Accessed 2013].

[42] BeagleBone, “What is BeagleBone?,” [Online]. Available: http://beagleboard.org/bone. [Accessed 2013].

[43] BeagleBoard.org, “What is BeagleBoard-xM?,” [Online]. Available: http://beagleboard.org/hardware-xm. [Accessed 2013].

[44] BeagleBoard.org, “BeagleBoard Product Details,” [Online]. Available: http://beagleboard.org/hardware. [Accessed 2013].

[45] R. Wies, “Policy definition and classification: aspect, criteria and examples,” Munich Network Management Team, University of Munich, Department of Computer Science, Munich.

[46] “Information Technology – Open Systems Interconnection – Systems Management Overview – Amendment 2: Management Domains Architecture,” PDAM 10040/2, ISO/IEC, November 1993.

[47] ““Information Technology – Open Systems Interconnection – SystemsManagement – Part 19: Management Domain and Management Policy Management Function”,” CD 10164-19, ISO/IEC, January 1994.

[48] ““Basic Reference Model of Open Distributed Processing – Part 1: Overview and Guide to Use”,” WD 10746-1, ISO/IEC, November 1993.

[49] M. Sloman, “Specifying Policy for Management of Distributed Systems,” IFIP, Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, October 1993.

[50] J. D. Moffett, “chapter 17,” in *Specification of Management Policies and Discretionary Access Control*, June 1994, p. 455–481.

[51] R. Wies, “Policies in Network and Systems Management – Formal Definition and Architecture,” *Manu Malek, editor, Journal of Network and Systems Management - Plenum Publishing Corporation,* vol. 2, pp. 63-83, March 1994.

[52] IDSM Project; SysMan Project, «Domain and Policy Service Specification,» IDSM Deliverable D6 / SysMan DeliverableMA2V2, IDSM Project (ESPRITIII EP 6311) and SysMan Project (ESPRIT III EP

7026), October 1993.

[53] J. D. Moffett and M. S. Sloman, “The Representation of Policies as System Objects,” in *In Conference on Organizational Computing Systems, SIGOIS Bulletin, pp 171–184*, Atlanta, November 1991.

[54] M. Porter and V. Millar, “How information gives you competitive advantage,” *Harvard Business Review,* vol. 4, no. 63, pp. 149-160, 1985.

[55] «Information Processing Systems – Open Systems Interconnection – Basic Reference Model,» IS 7498, ISO/IEC, 1984.

[56] M. Masullo and S. Calo, “Policy Management: An Architecture and Approach,” Proceedings of the IEEE First International Workshop On Systems Management, Los Angeles, April 1993.

[57] “Information Technology – Open Systems Interconnection – Systems Management – Management Functions,” IS 10164-X, ISO/IEC.

[58] Open Software Foundation, “OSF Distributed Management Environment (DME) Architecture,” 1992.

[59] Object Management Group, “Object Management Architecture Guide,” Document 92-11-1, September 1992.

[60] Object Management Group, “Object Services Architecture,” Document 92-8-4, August 1992.

[61] J. McLean, “A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions,” in *Proceedings of the 1994 IEEE Symposium on Security and Privacy, pp. 79-93*, IEEE Press, May 1994.

[62] D. Mccullough, “Noninterference and the composability of security properties,” *Security and Privacy Proceedings, IEEE Symposium (http://doi: 10.1109/SECPRI.1988.8110),* pp. 177-186, 1988.

[63] A. Zakinthinos, “On the composition of security properties,” PhD Thesis. University of Toronto, Toronto, 1996.

[64] A. Zakinthinos and E. Lee, “A general theory of security properties,” *Security and Privacy Proceedings, IEEE Symposium (http://doi: 10.1109/SECPRI.1997.601322),* pp. 94-102, May 1997.

[65] H. Mantel, “On the composition of secure systems,” *Security and Privacy Proceedings, IEEE Symposium (http://doi: 10.1109/SECPRI.2002.1004364),* pp. 88-101, 2002.

[66] K. Trivedi, D. S. Kim, A. Roy and D. Medhi, “Dependability and security models,” *Design of Reliable Communication Network (DRCN 2009), 7th International Workshop (http://doi: 10.1109/DRCN.2009.5340029),* pp. 11-20, 25-28 Oct. 2009.

[67] D. Nicol, W. Sanders and K. Trivedi, “Model-based evaluation: from dependability to security,” *Dependable and Secure Computing, IEEE Transactions (http://doi: 10.1109/TDSC.2004.11),* vol. 1, no. 1, pp. 48-65, Jan - March 2004.

[68] Sallhammar, Karin, B. E. Helvik and S. J. Knapskog, “On Stochastic Modelling for Integrated Security and Dependability Evaluation,” *Journal of Networks (Web. 14 Feb. 2013),* pp. 31-42, 2006.

[69] E. Jonsson, “Towards an integrated conceptual model of security and dependability,” *Availability, Reliability and Security (ARES 2006). The First International Conference (http://doi: 10.1109/ARES.2006.138),* pp. 8; 20-22, 2006.

[70] M. Malhotra and K. Trivedi, “Dependability modelling using Petri-nets,” *Reliability, IEEE Transactions (http://doi: 10.1109/24.406578),* vol. 44, no. 3, pp. 428-440, Sep 1995.

[71] J. L. Peterson, “Petri Nets. Computing Surveys,” *http://doi.acm.org/10.1145/356698.356702,* vol. 9, no. 3, pp. 223-252, September 1977.

[72] J. L. Peterson, Petri Net Theory and the Modelling of Systems. Translated into Russian and Japanese. ISBN 0-13-661983-5, Prentice-Hall, Englewood Cliffs: New Jersey, April 1981.

[73] J. Wang, Petri nets for dynamic event-driven system modelling (Handbook of Dynamic System Modeling), Paul Fishwick, CRC Press, 2007.

[74] E. Jonsson, “An integrated framework for security and dependability,” in *In Proceedings of the 1998 workshop on New security paradigms (NSPW '98), pp. 22-29 (DOI=10.1145/310889.310903 http://doi.acm.org/10.1145/310889.310903)*, ACM, New York, USA, 1998.

[75] C. Cassandras and S. Lafortune, Introduction to Discrete Event Systems - Second Edition, Springer - ISBN 978-0-387-33332-8. (771+xxiii pages), 2008, pp. ISBN 978-0-387-33332-8. (771+xxiii pages).

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

[77] C.A. Petri, Kommunikation mit Automaten, “New York: Griffiss Air Force Base, Technical Report RADC-TR-65--377,” *Institut für Instrumentelle Mathematik, Schriften des IIM,* vol. 1, no. 2, p. Suppl. 1 (english translation), 1966.

[78] 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 Perspectives - Lecture Notes in Computer Science, Springer,* no. 803, pp. 230-272, 1994.

[79] K. Jensen, “A Brief Introduction to Coloured Petri Nets, in E Brinksma (ed.),” *Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science - Springer,* no. 127, pp. 203-208, 1997.

[80] K. Jensen and L. Kristensen, “Coloured Petri Nets: Modelling and Validation of Concurrent Systems,” *Springer,* p. 384, 2009.

[81] T. Murata, “Petri Nets: Properties, Analysis and Applications,” *invited survey paper, Proceedings of the IEEE (http://www.cs.uic.edu/~murata/PAPERs/1989.IEEE.Proc.pdf.gz),* vol. 77, no. 4, pp. 541-580, April, 1989.

[82] F. Commoner, “Deadlocks in Petri nets,” Applied Data Research Inc., Wakefield, MA 1972.

[83] S. Singh, M. Cukier and W. Sanders, “Probabilistic validation of an intrusion-tolerant replication system,” in *In de Bakker, J.W., de Roever, W. P., and Rozenberg, G., editors, International Conference on Dependable Systems and Networks (DSN‘03)*, June 2003.

[84] D. Wang, B. Madan and K. Trivedi, “Security Analysis of SITAR Intrusion Tolerance System,” in *ACM SSRS’03*, 2003.

[85] S. H. Houmb and K. Sallhammar, “Modelling System Integrity of a Security Critical System using Coloured Petri Nets,” in *In Proceedings of the 1st International Conference on Safety and Security Engineering (SAFE 2005)*, Rome, Italy, June 13-15, 2005.

[86] J. L. Rasmussen and M. Singh, “Designing a Security System by Means of Coloured Petri Nets,” in *In Proceedings of the 17th International Conference on Application and Theory of Petri Nets, pp. 400-*

419, Jonathan Billington and Wolfgang Reisig (Eds.). Springer-Verlag,, London, UK, 1996.

[87] R. D. R. Bouroulet, H. Klaudel, E. Pelz and F. Pommereau, “Modelling and analysis of security protocols using role based specifications and Petri nets,” in *Proceedings of ICATPN'08, LNCS 5062, Springer*, 2008.

[88] F. Pommereau, Algebras of coloured Petri nets, and their applications to modelling and verification, LAP LAMBERT Academic Publishing (ISBN: 978-3-8433-6113-2), 2010.

[89] W. Dresp, “Security analysis of the secure authentication protocol by means of coloured petri nets. (http://dx.doi.org/10.1007/11552055_23),” *[J. Dittmann, S. Katzenbeisser,A. Uhl (Eds.)] Springer-Verlag, Berlin, Heidelberg,* pp. 230-239, 2005.

[90] K. Juszczyszyn, “Verifying enterprise's mandatory access control policies with coloured Petri nets (doi: 10.1109/ENABL.2003.1231405),” *Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '03). Proceedings Twelfth IEEE International Workshops on,* pp. 184-189, June 2003.

[91] B. Mikolajczak and S. Joshi, “Modelling of information systems security features with colored Petri nets,” *Systems, Man and Cybernetics, 2004 IEEE International Conference on (http://doi: 10.1109/ICSMC.2004.1401304),* vol. 5, pp. 4879-4884, 2004.

[92] “Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management,” in *IFIP*, October 1993.