Project no: 269317

nSHIELD

new embedded Systems arcHItecturE for multi-Layer Dependable solutions

Instrument type: Collaborative Project, JTI-CP-ARTEMIS

Priority name: Embedded Systems

D5.2: Preliminary SPD Middleware and Overlay Technologies Prototype

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	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document Authors and Approvals

Authors		Date	Signature
Name	**Company**		
Andrea Morgagni	SES		
Andrea Fiaschetti	UNIROMA1		
Balázs Berkes	S-LAB		
Nikos Pappas	HAI		
Kostas Rantos	TUC		
Kostas Fysarakis	TUC		
Alexandros Papanikolaou	TUC		
Dimitris Geneiatakis	TUC		
Harry Manifavas	TUC		
Silvano Mignanti	UNIROMA1		
Andrea Lanna	UNIROMA1		
Gaetano Scarano	UNIROMA1		
Roberto Cusani	UNIROMA1		
Martina Panfili	UNIROMA1		
Vincenzo Suraci	UNIROMA1		
Francesco Delli Priscoli	UNIROMA1		
Panagiotis Soufrilas	ATHENA		
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		
AD3		
AD4		
AD5		
AD6		

Modification History		
Issue	**Date**	**Description**
Draft A	07/01/2013	First ToC emission
Draft B	10/05/2013	First inputs
Final	30/06/2013	All

Executive Summary

This document includes the preliminary Middleware & Overlay prototypes for the nSHIELD project and represents the main output of the first development phase. As declared in the Technical Annex, these prototypes are heterogeneous in shape and purpose, and their common objective is to provide the Embedded System and Software domains with enriched technologies that hopefully could be integrated into a common platform towards a demonstrator.

The document's structure (for the available prototypes) is similar to D5.3, and in particular:

1. Introduction: a brief introduction to the document contents
2. Semantic technologies prototypes
3. Core serviced at Middleware level prototypes
4. SHIELD Policy Based Management prototypes
5. Overlay prototypes
6. Brief conclusions
7. 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 SHIELD Middleware provides a set of innovative technologies to implement security functionalities as well as composition for SPD purposes. In this document the following prototypes are presented:

Table 1-1: Prototype List

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

2 Semantic Technologies Prototypes

2.1 OWL/ER Diagrams of the SHIELD semantic model

The OWL and E-R Diagrams for the SHIELD semantic models are reported below and attached in the zip file **UNIROMA1_SHIELD_Semantic_Models.zip**

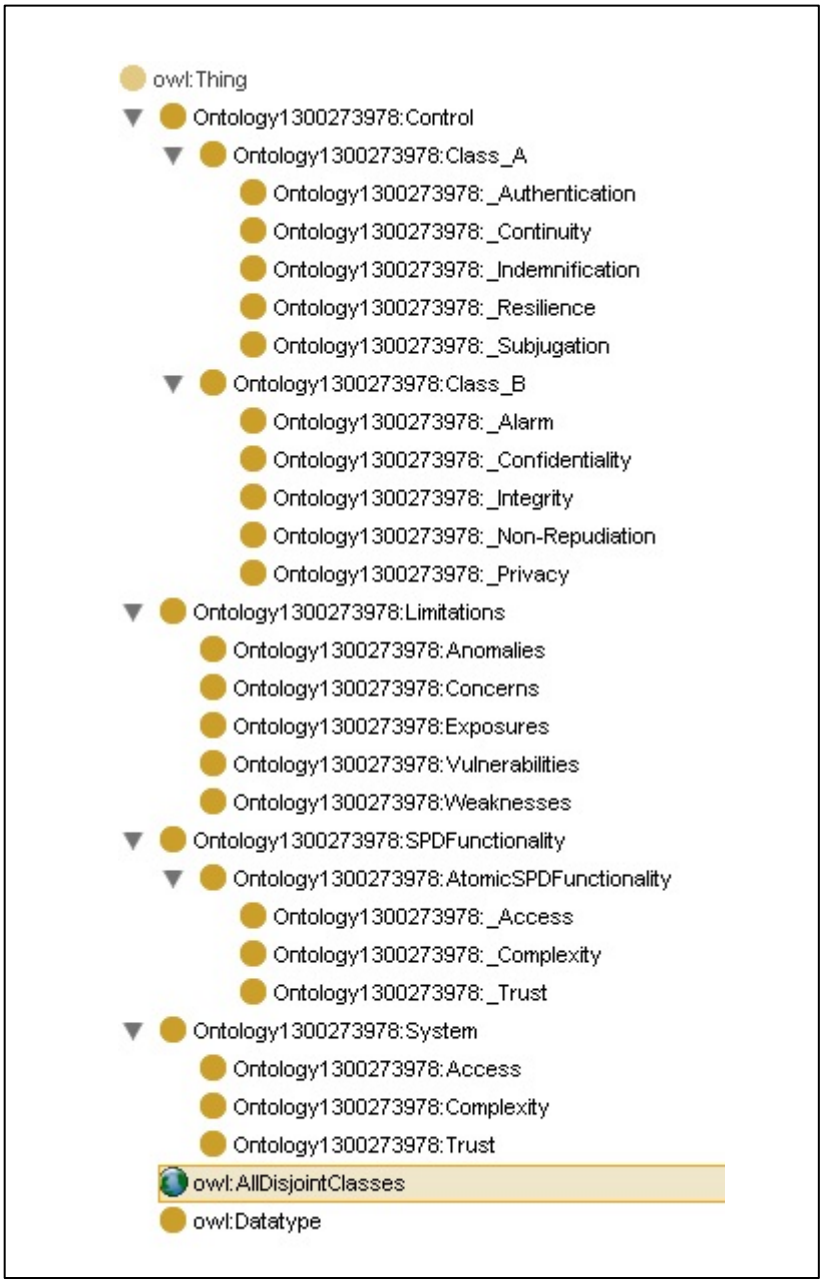

Figure 2-1: SHIELD OWL

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY protege "http://protege.stanford.edu/plugins/owl/protege#" >
    <!ENTITY xsp "http://www.owl-ontologies.com/2005/08/07/xsp.owl#" >
    <!ENTITY TCP "http://www.owl-ontologies.com/Ontology1300273978.owl#TCP/" >
]>


<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1300273978.owl#"
     xml:base="http://www.owl-ontologies.com/Ontology1300273978.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:TCP="http://www.owl-ontologies.com/Ontology1300273978.owl#TCP/">
    <owl:Ontology rdf:about="">
        <owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/protege"/>
    </owl:Ontology>
    <owl:Class rdf:ID="_Access">
        <rdfs:subClassOf rdf:resource="#AtomicSPDFunctionality"/>
    </owl:Class>
    <owl:Class rdf:ID="_Alarm">
        <rdfs:subClassOf rdf:resource="#Class_B"/>
    </owl:Class>
    <owl:Class rdf:ID="_Authentication">
        <rdfs:subClassOf rdf:resource="#Class_A"/>
    </owl:Class>
    <owl:Class rdf:ID="_Complexity">
        <rdfs:subClassOf rdf:resource="#AtomicSPDFunctionality"/>
    </owl:Class>
    <owl:Class rdf:ID="_Confidentiality">
        <rdfs:subClassOf rdf:resource="#Class_B"/>
    </owl:Class>
    <owl:Class rdf:ID="_Continuity">
        <rdfs:subClassOf rdf:resource="#Class_A"/>
    </owl:Class>
    <owl:Class rdf:ID="_Indemnification">
        <rdfs:subClassOf rdf:resource="#Class_A"/>
    </owl:Class>
    <owl:Class rdf:ID="_Integrity">
        <rdfs:subClassOf rdf:resource="#Class_B"/>
    </owl:Class>
    <owl:Class rdf:ID="_Non-Repudiation">
        <rdfs:subClassOf rdf:resource="#Class_B"/>
    </owl:Class>
    <owl:Class rdf:ID="_Privacy">
        <rdfs:subClassOf rdf:resource="#Class_B"/>
    </owl:Class>
    <owl:Class rdf:ID="_Resilience">
        <rdfs:subClassOf rdf:resource="#Class_A"/>
```

```
    </owl:Class>
    <owl:Class rdf:ID="_Subjugation">
        <rdfs:subClassOf rdf:resource="#Class_A"/>
    </owl:Class>
    <owl:Class rdf:ID="_Trust">
        <rdfs:subClassOf rdf:resource="#AtomicSPDFunctionality"/>
    </owl:Class>
    <owl:Class rdf:ID="Access">
        <rdfs:subClassOf rdf:resource="#System"/>
    </owl:Class>
    <owl:Class rdf:ID="Anomalies">
        <rdfs:subClassOf rdf:resource="#Limitations"/>
    </owl:Class>
    <owl:Class rdf:ID="AtomicSPDFunctionality">
        <rdfs:subClassOf rdf:resource="#SPDFunctionality"/>
        <rdfs:subClassOf>
    <owl:Class rdf:ID="Class_A">
        <rdfs:subClassOf rdf:resource="#Control"/>
    </owl:Class>
    <owl:Class rdf:ID="Class_B">
        <rdfs:subClassOf rdf:resource="#Control"/>
    </owl:Class>
    <owl:Class rdf:ID="Complexity">
        <rdfs:subClassOf rdf:resource="#System"/>
    </owl:Class>
    <owl:Class rdf:ID="Concerns">
        <rdfs:subClassOf rdf:resource="#Limitations"/>
    </owl:Class>
        <owl:Class rdf:ID="Control"/>
        </owl:ObjectProperty>
        <owl:Class rdf:ID="Exposures">
            <rdfs:subClassOf rdf:resource="#Limitations"/>
        </owl:Class>
                        <owl:Class rdf:ID="Trust">
                            <rdfs:subClassOf rdf:resource="#System"/>
                        </owl:Class>
                            <owl:Class rdf:ID="Vulnerabilities">
                                <rdfs:subClassOf rdf:resource="#Limitations"/>
                            </owl:Class>
                            <owl:Class rdf:ID="Weaknesses">
                                <rdfs:subClassOf rdf:resource="#Limitations"/>
                            </owl:Class>
                            <rdf:Description
</rdf:RDF>
```

Figure 2-2: SHIELD XML

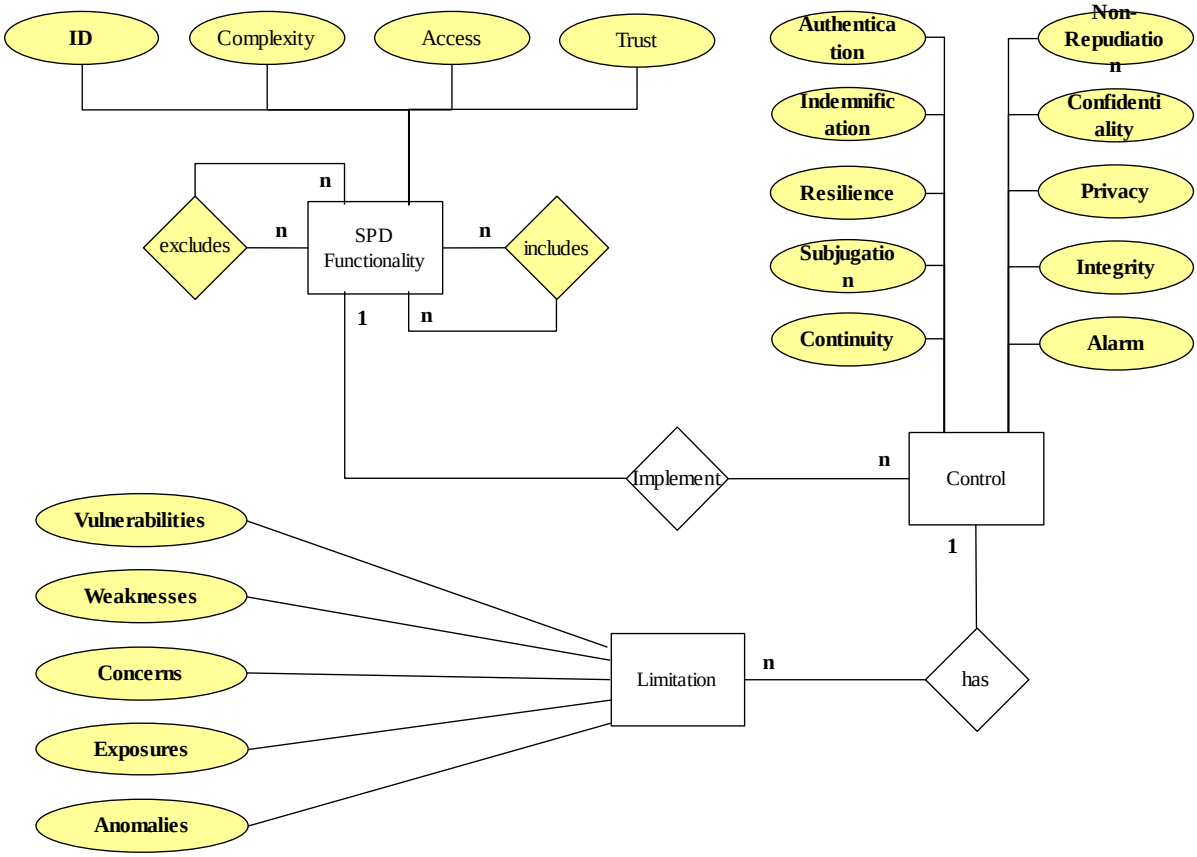

Figure 2-3: Domain Dependent Library E-R

2.2 Ontology for Intrusion Detection System

The Ontology for Intrusion Detection System is included in the zip file **MGEP_SHIELD_IDS_Ontology.zip**

The source code is not reported due to the excessive length (about 100 pages).

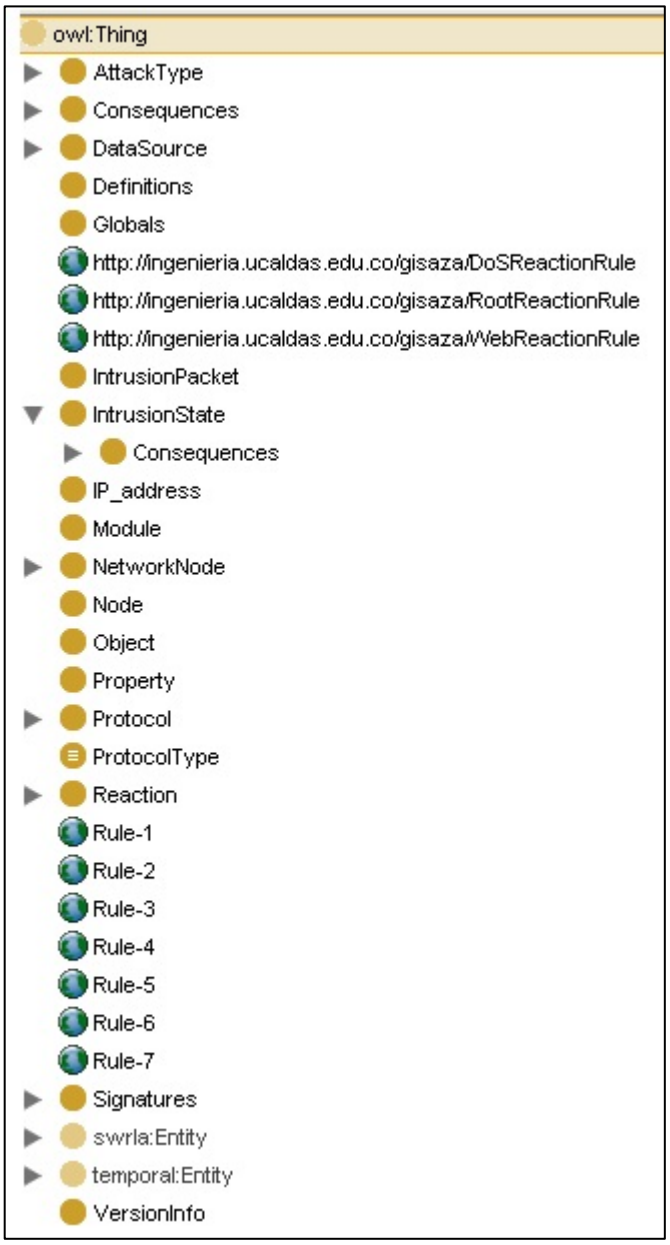

Figure 2-4: OWL for IDS

3 Core Services at Middleware Level Prototypes

3.1 Protocol for Secure Discovery

The Prototype of the SHIELD Secure Service Discovery in included in the zip file **UNIROMA1_SHIELD_Secure_Service_Discovery.zip**.

While the description of the OSGI bundle is reported in D5.3, in the following some practical results about the prototype tests are presented.

3.1.1 Test

To test and verify the Secure Service Discovery implemented, we have implemented an open source PKI called *OpenCA*. Once done it, in particular once we have created a Certification Authority and a Registration Authority, we set all system generating the two key.

From the point of view of the SLP, we have used a tool called *SLP Daemon*. This latter already implements a user interface to monitor the exchange for of messages and the other activities. To try the Secure Service Discovery we have implemented a new bundle java.

3.1.1.1 Service Registration and Secure Service Registration

The first test consists in a request from a UA to a DA using the traditional SLP *Service Registration* message including the digital signature. The test included the sending and the verifying of the signature and the message. In the screenshot in Figure 3-1, we can see the log of the registration of service "*service:http//www.nosecure.it*".

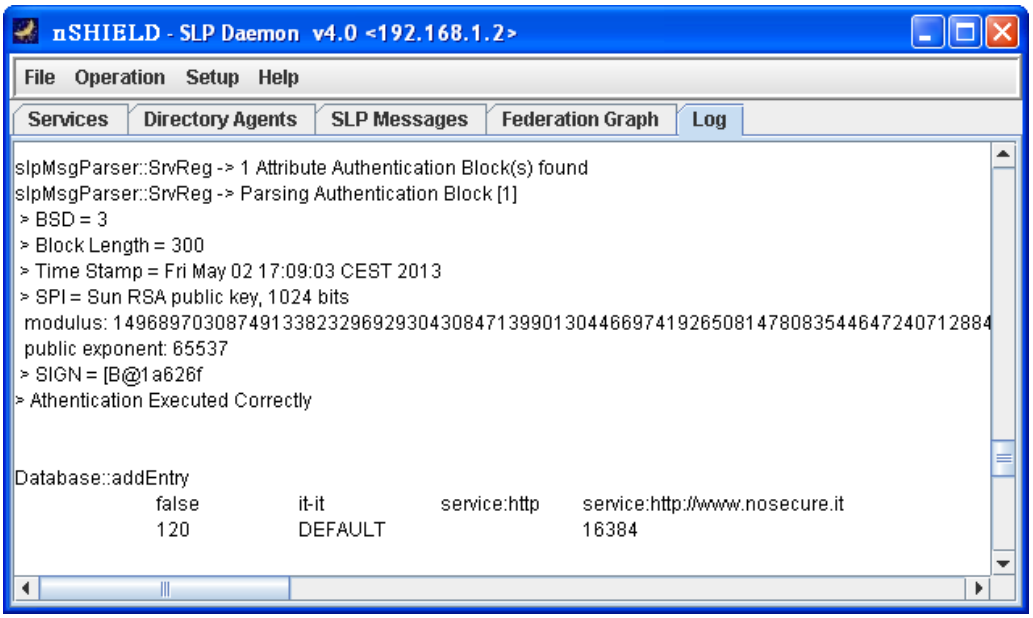

Figure 3-1: Service Registration test.

The follow figure (Figure 3-2) shows how happens when a request arrives using a *Secure Service Discovery*. In this case we have required registering a service called "*service:http//www.secure.it*". Note the attribute "*authentication=true*". It identifies the necessary of protected communication.

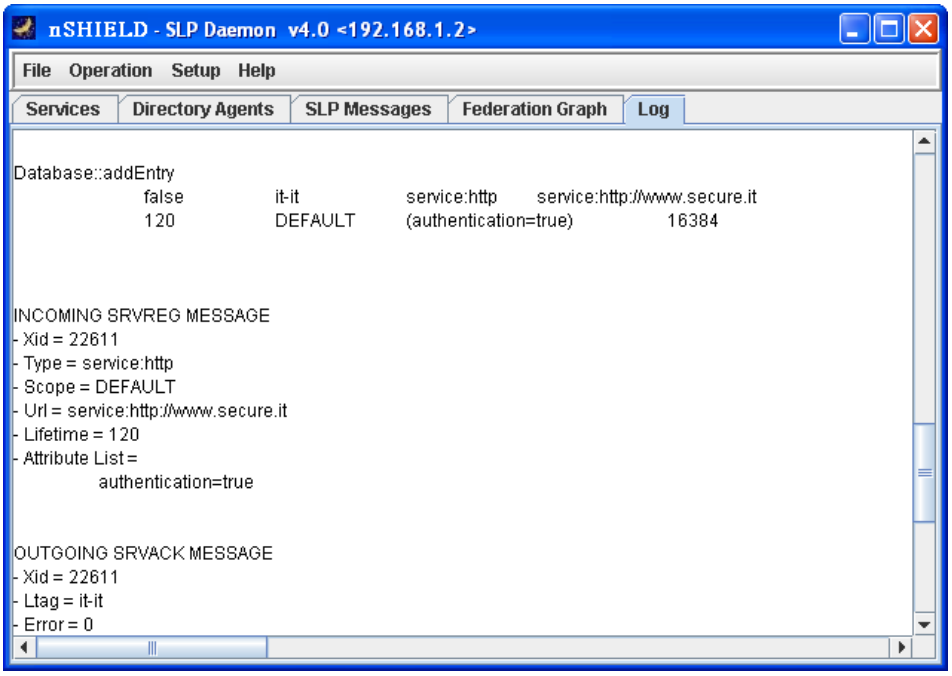

Figure 3-2: Secure Service Registration test

3.1.1.2 Service Request and Secure Service Request

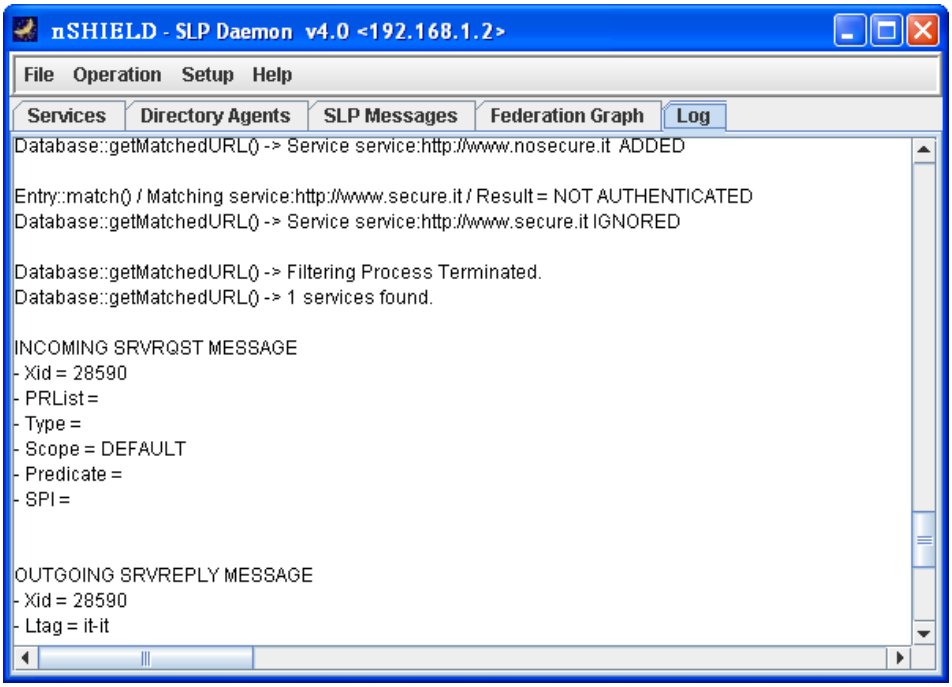

Figure 3-3: traditional Service Request test

In response at Service Registration and at Secure Service Registration messages, the DA reply through the Service Replay and the Secure Service Replay, respectively,

If the registration was been in traditional or protected mode, the services visible are different. Using traditional request, the UA could see only services for which do not require the authentication. The Figure 3-3 shows an example.

The follow screenshot shows, on the contrary, a request when the UA use protected messages. The UA receives all information of whole system. The field *ErrorCode* of the *Replay* message indicates what error happens (if one occurs).

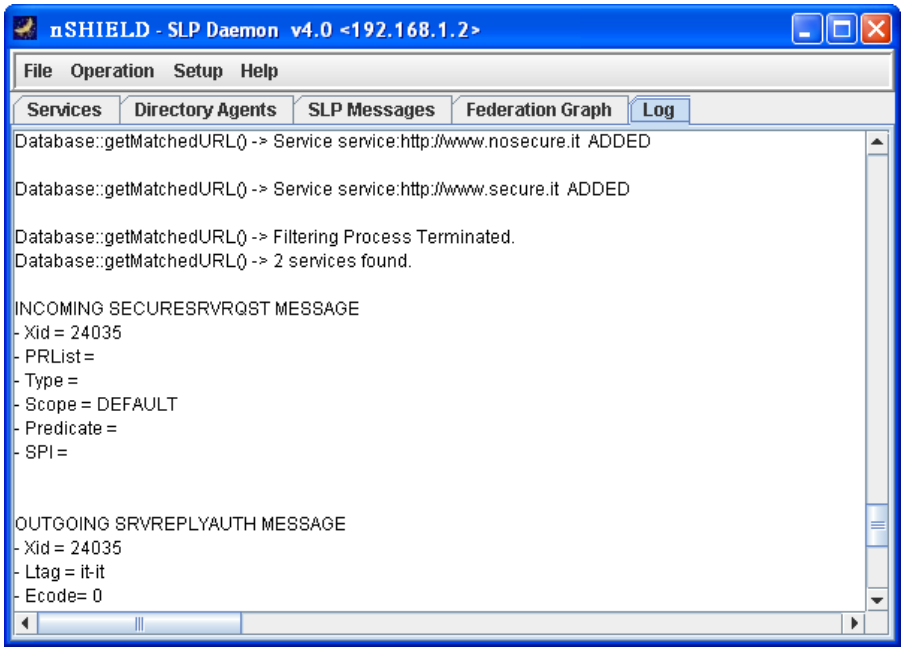

Figure 3-4: Secure Service Request test

3.2 Monitoring, filtering and intrusion detection module

3.2.1 Module implementation

3.2.1.1 Filtering and Intrusion Detection Bundle

The latest version of the Filtering and Intrusion Detection Bundle is available at the SVN for WP5, maintained by UNIROMA1, as well as in the attached **SLAB_SHIELD_IDS.zip** file.

The module implementation contains four major parts, from which the first two constitutes the necessary functionality for the module; the latter two provides development and testing tools and examples:

Table 3-1: Parts of the module "*Monitoring, filtering and intrusion detection*"

Code part	Description
CounterDoS	DoS protection module (so/DLL/EXE) source with JNI wrapper for DLL
CounterDoSJ	Java wrapper class (with JNI interface to module) Java classes for services and collection of services OSGI Bundle Activator class for Filtering and Intrusion Detection Bundle
CounterDoSJDemo	Java Demonstration class: a Bundle that utilizes DoS protection service
CounterDoSJTest	Java test classes: verification test cases for main features of module

Please see README.TXT in respective code parts for description on compilation and usage.

3.3 Adaptation of legacy systems

The source code of this prototype is included in the attached file **ATHENA_SHIELD_Adaptation_of_Legacy_Systems.zip**.

3.3.1 Service Location Protocol

For our deployment environmet we use the Knopflerfish framework and its plugin for Eclipse IDE.The Knopflerfish Eclipse Plug-in is a tool for launching and debugging the Knopflerfish OSGi distribution. The goal with the plug-in is to simply the use of Knopflerfish for developers using Eclipse as their IDE.

The basic configurations for our bundles is to use the R-OSGi,jslp-osgi ,R-OSGi SLP Service Discovery services by importing there packages in our bundles.

- jslp-osgi-1.0.0.RC5.jar
- remote-1.0.0.RC4.jar
- service_discovery.slp-1.0.0.RC4.jar

Both the client (Legacy nodes) and Server sides (nSHIELD-GW) have to import to their bundles the R-OSGi service.The client side has to additonaly import to its bundles the jslp-osgi , R-OSGi SLP Service Discovery services in order to access the remote dervices provided by the nSHIELD-GW.

The ad-hoc software of the server side are bundles that register to R-OSGi the nSHIELD services in order to make them visible outside.The ad-hoc software of the client side are bundles that connect to a GW and get the service.

For our example to show the implementation of such scenario we created a very simple service Nservice (Nshield service) that runs in server side and registers itself to R-OSGi.On the other hand the client side runs a LeNoReSer (Legacy Node Service) that connects remotely to the server and gets access to the service.

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

```
package Nservice;

import java.util.Dictionary;
import java.util.HashMap;
import java.util.Map;
import java.util.Hashtable;
import java.util.Enumeration;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

import ch.ethz.iks.r_osgi.RemoteOSGiService;


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(Nservice.class.getName(), new Nservice(), properties);
    }

  public void stop(BundleContext bundleContext)
  {
    System.out.println("Hello stopped.");
  }
}
```

Now, other R-OSGi enabled peers can connect to the peer and get access to the service.

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

```
package LeNoReSer;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.BundleException;
import org.osgi.framework.ServiceReference;

import Nservice.Nservice;

import ch.ethz.iks.r_osgi.RemoteOSGiService;
import ch.ethz.iks.r_osgi.RemoteServiceReference;
import ch.ethz.iks.r_osgi.URI;

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://150.140.xxx.xxx:9278"));
         final RemoteServiceReference[] srefs = remote.getRemoteServiceReferences(new URI("r-
osgi://150.140.xxx.xxx:9278"), Nservice.class.getName(), null );


Nservice hi = (Nservice) remote.getRemoteService(srefs[0]);
 hi.Echo();*/
  }

  /* (non-Javadoc)
   * @see org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
   */
```

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

With the call of the getRemoteService method, a local proxy for the remote service is created. The service proxy is registered with the local service registry and can also be retrieved like a normal OSGi service. To get rid of a remote service, you can call ungetRemoteService.

3.4 Middleware Protection profile

The middleware protection profile is attached in the file **SES_SHIELD_Middleware_PP_v1.0.zip**

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

4 SHIELD policy based access control

4.1 SHIELD policy based access control architecture

While a Policy Based Access Control description is reported in the following, the source code for some implementations is included in the zip file **TUC_SHIELD_Policy_Based_Access_Control.zip**.

4.1.1 Description

The SHIELD policy based access control architecture targets heterogeneous embedded systems and features provisions for interoperability with existing standards, facilitating communication over diverse networks. The proposed framework is DPWS-compliant, utilizing the relevant specifications and existing work to provide message-level security and fine-grained security policy functionality while maintaining interoperability with the standard. It consists of several components that run on different nodes of the nShield architecture. These components are the Policy Enforcement Points (PEP), the Policy Administration Point (PAP), the Policy Decision Points (PDP) and the Policy Information Point (PIP), the interconnection of which can be seen in Figure 4-1.

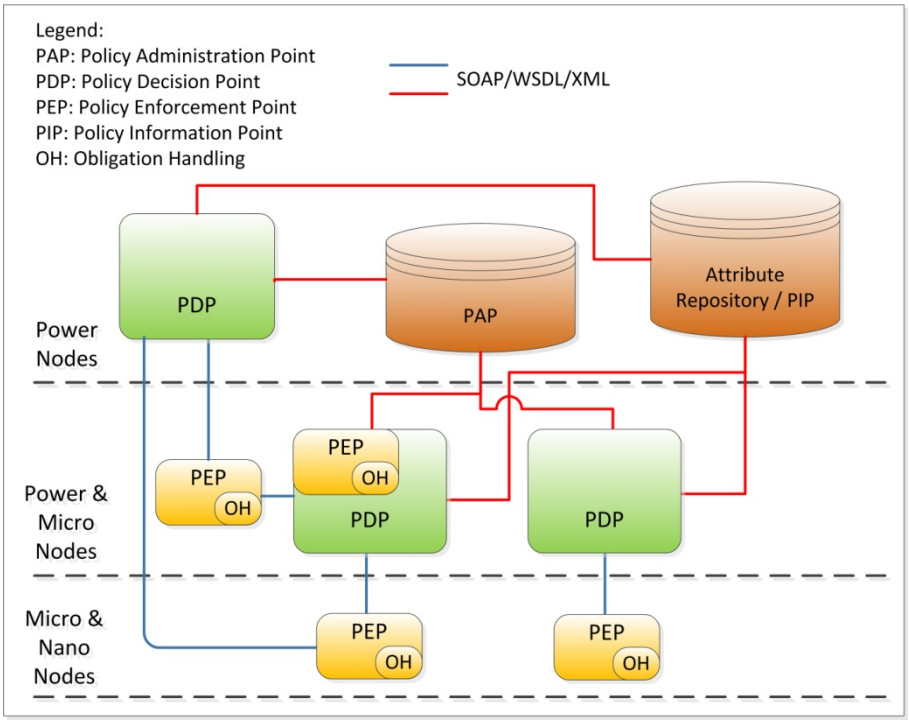

Figure 4-1. Policy-based architecture

A node, depending on its capabilities and the available resources, might include one or more of these functional components. The policy-based management prototypes utilize the technologies listed below.

I. Nano node

a) Role

i) DPWS Device hosting services and their operations.

ii) Policy Enforcement Point

b) Underlying technologies
- i) Operating system: Contiki
- ii) Network: 802.15.4/6LoWPAN
- iii) DPWS platform: uDPWS stack (C)

c) Prototype platforms
- i) Zolertia Z1 motes
  - (1) http://www.zolertia.com/ti

II. Micro node

a) Role
- i) DPWS Device hosting services and their operations.
- ii) Policy Enforcement Point
- iii) Bridge between 802.15.4/6LoWPAN and IPv4/IPv6 networks (optional)

b) Underlying technologies
- i) **Operating system:** A lightweight Linux distribution
- ii) **Network:** 802.15.4/6LoWPAN (optional), IPv4/IPv6
- iii) Bridge between 802.15.4/6LoWPAN and IPv4/IPv6 networks (optional)
- iv) DPWS platform: WS4D-gSOAP (C)

c) Prototype platforms
- i) Beaglebone
  - (1) http://beagleboard.org/bone

III. Power node

a) Role
- i) DPWS peer (device & client)
- ii) Responsible for interfacing with OSGi (Knopflerfish) framework.
- iii) Bridge between 802.15.4/6LoWPAN and IPv4/IPv6 networks
- iv) Policy Administration Point
- v) Policy Information Point – Policy Administration Point

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

c) Prototype platforms
- i) Beagleboard xM
  - (1) http://beagleboard.org/hardware-xm
- ii) Beagleboard
  - (1) http://beagleboard.org/hardware

4.2 Policy Definition

The following example on policy definition and implementation are taken from D5.3

4.2.1 Policy examples

4.2.1.1 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*

Analyzing 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.

Example 2:

”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.

4.2.1.2 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 Overlay Prototypes

5.1 Security Agent Implementation

Apart from the definition of proper control algorithms, a significant effort has been put to design and implement the structure of the Bundle that implement the behaviour of the Security Agent as defined in the Architecture document. The basic architecture is reported in the following, while the bundle is included in the zip file **UNIROMA1_SHIELD_Security_Agent.zip**.

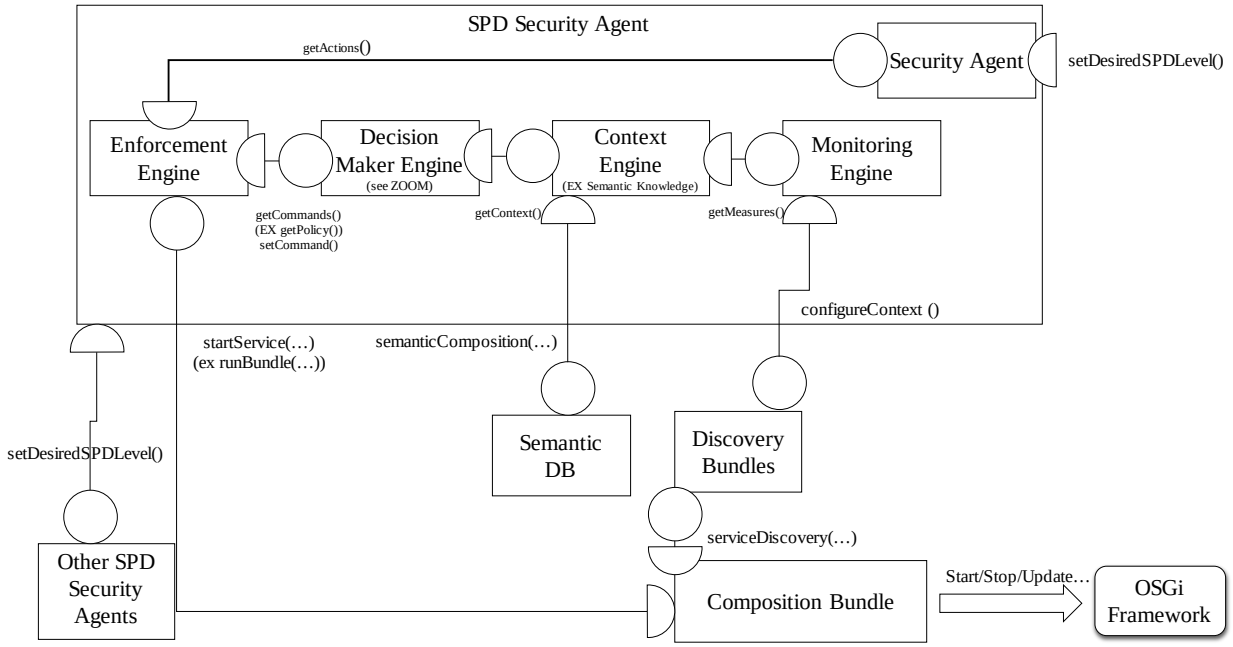

Figure 5-1: Security Agent Bundle structure

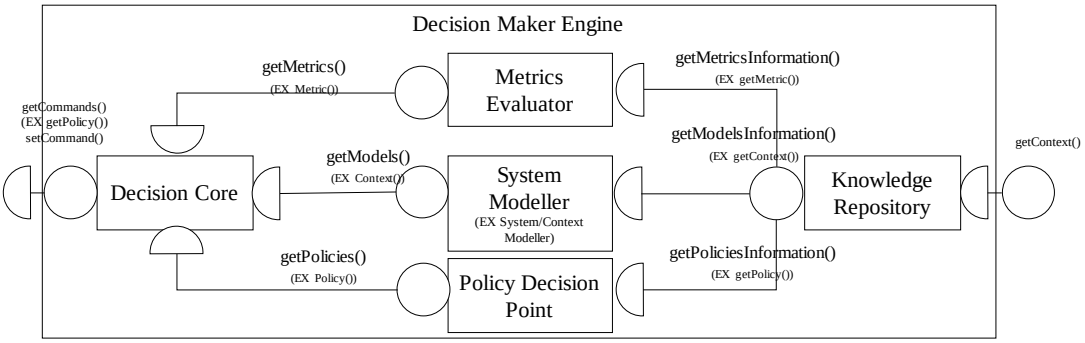

Figure 5-2: Decision Maker Engine Bundle structure

5.2 Coloured Petri Nets (CPN) composition algorithms

The source code of the prototype described in the following is available in the attachment **UNIROMA1_SHIELD_Petri_Net_Models.zip**

5.2.1 CPN tools model

The model of nSHIELD system is based on Coloured Petri Net formalism as described in the nSHIELD *Deliverable 5.3: Preliminary SPD middleware & Overlay technologies prototype Report*. The Coloured Petri Net, that represents the system, is edited and simulated with the CPN Tools ([1], [2]) available on line at http://cpntools.org/.

The model of nSHIELD system has a single page *Functionalities*. This page has *N*+*M* transitions which all are substitution transitions and represent respectively *N* SPD Functionalities and *M* type of relation/constraint between SPD functionalities. For the sake of simplicity, in this document, we consider only two SPD functionalities and only two types of relation (*N* = *M* = 2). In particular, we consider the following SDP functionalities: *authentication* and *identification*, and the following types of relation: *coupling* and *mutual exclusion.*

A CPN model is usually created as a graphical drawing; in the Figure 5-3 the basic CPN model of the nSHIELD system is shown.

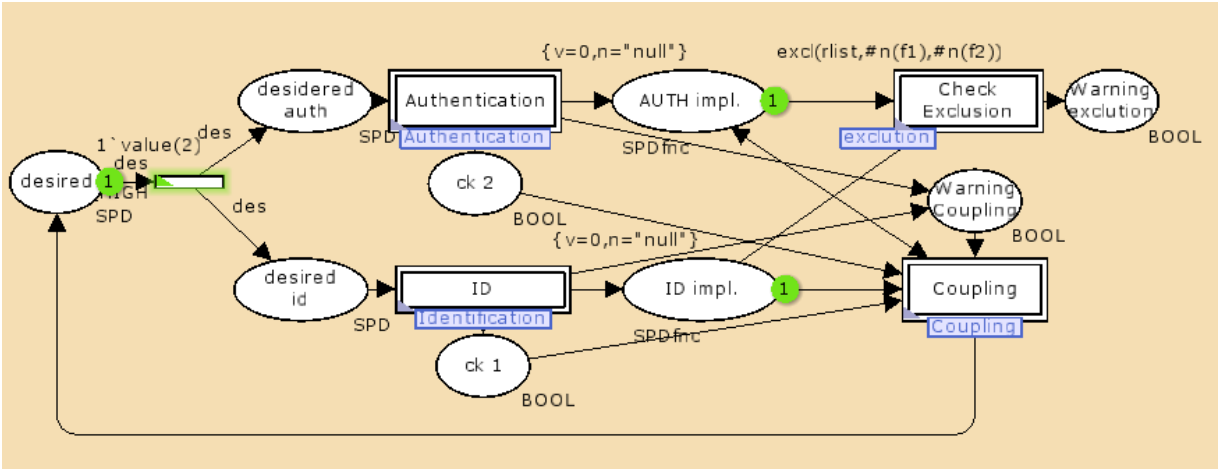

Figure 5-3: nSHIELD CPN model in the initial marking

The CPN model, in Figure 5-3, contains: i) nine places (drawn as ellipses), ii) four transitions (drawn as rectangular boxes), iii) a number of directed arcs connecting places and transitions, and iv) textual inscriptions next to the places, transitions, and arcs. CPN tools are based on CPN ML programming language an extension of the Standard ML language. Places and transitions (the nodes) with the directed arcs constitute the net structure. Note that an arc is always a connection place-transition or transition-place ([2]).

As described in nSHIELD Deliverable D5.3, the system state is represented by the places. In particular each place can be marked with one or more tokens, and each token can carry a data value called token colour. The number of tokens in each place, together with the associated token colours represents the state of the system and is called the marking of the CPN model. Otherwise the place marking indicates the tokens on a precise place. Then in our model the state of each SPD functionality nSHIELD system is modelled by the place *Desired* (level of SPD desired), one place *Implemented* for each functionality and

one place *Warning* for each type of relation/constraint. The tokens on each place can be assumed in a determinate set called colour set of the place. By convention the colour set is written near the place. The colour set of all the places in our model are defined in Figure 5-4.

```
Color Sets
  colset SPDvalue = int with 0..100;
  colset INT= int;
  colset SPDname = string with "A".."z" and 2..5;
  colset BOOL =bool;
  colset SPDfnc = record v:SPDvalue*n:SPDname;
  colset SPD = union name:SPDname + value:SPDvalue;
  colset REL = record n1:SPDname*n2:SPDname;
  colset RELlist = list REL;
  colset SPDlist = list SPDfnc;
```

Figure 5-4: The definition of the colour sets in the nSHIELD model

Considering, for example, the three places *Desired*, with the same colour set, SPD, defined as a union of colour set SPDvalue (a limited set of integers) and colour set SPDname (a limited set of strings formed by, at least two - up to five, alphabetic characters). In these places the token can be assume either an integer value in the interval [0,100], that represent the desired SPD level or a string with at least two - up to five, alphabetic characters that indicates a particular functionality implementation.

The places *Functionality* Implemented have the colour set SPDfnc, defined to be the set of Cartesian product of the values in SPDvalue and SPDname colour sets (note that each component is identified by a unique label so that each field is position-independent). The SPDfnc colour set is used to identify the functionality implementation, with its name and the corresponding SPD value (depending by the metric). The remaining places have the colour set BOOL, defined to be the set of Boolean values ({`true, false`}) `bool`.

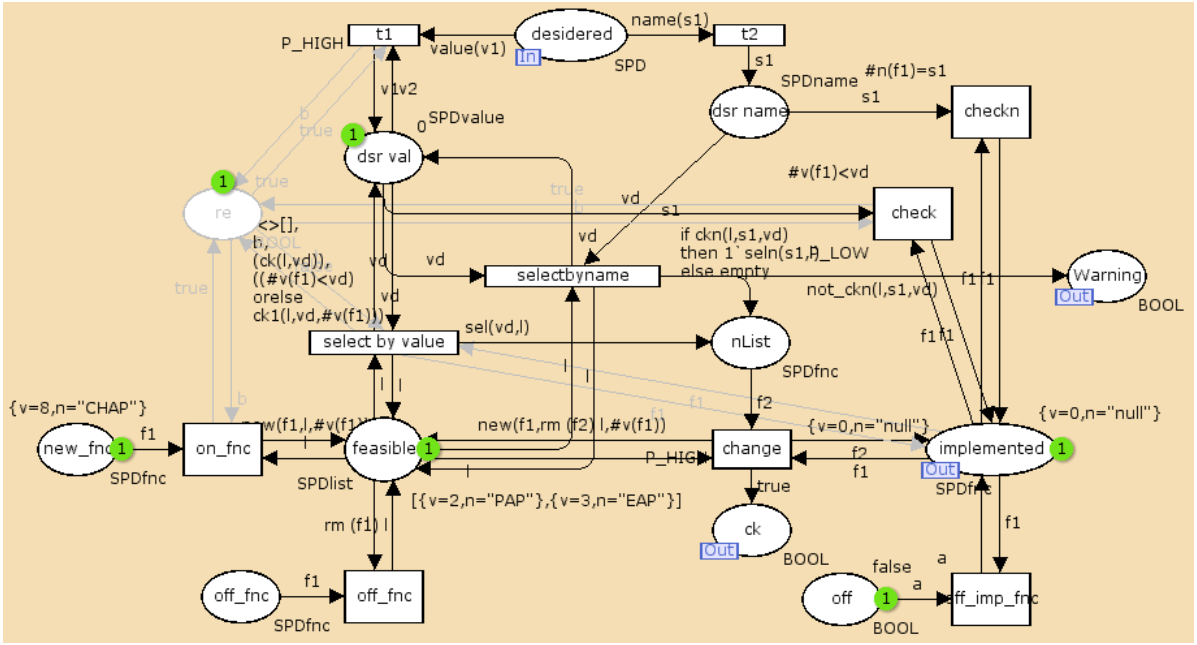

Figure 5-5: The SPD functionality sub-module

In Figure 5-3, next to places *desired*, *AUTH implementation* and *ID implementation*, it is possible to see, respectively, the inscriptions: `1`value(2)`, `{v=0,n="null"}` and `{v=0,n="null"}`, that specifies the initial marking of these places. This initial marking M_0 indicates that that the initial state of the system is: i) no functionality enabled and ii) desired SPD value equal to 2.

As explained previously our CPN model is organised as a set of hierarchically related modules. The main feature of hierarchical structure is the association of a sub-module with a substitution transition (in CPN Tools, substitution transitions can be recognised by the double boxes and with a rectangular sub-module tag). Intuitively, the sub-module, extended in a new page, presents a more detailed view of the behaviour represented by the substitution transition.

The Figure 5-5 shows the SPD functionality module. The structure of this module is the same for all SDP functionalities, while the token colours in the several places could be different.

The SPD functionality sub-module contains several transitions and place. For the sake of simplicity we will limit the description to the relevant nodes. The place *desired* is an input port, the places *implemented*; *ck* and *warning* are three output ports (in CPN Tools, port places can be recognised by the rectangular port-type tags). These places constitute the interface through which the SPD Functionality module exchanges tokens with its environment (i.e., the other modules). On the other hand, in the main page, the input/output places of substitution transitions, called input/output sockets, constitute the interface of the substitution transition. To complete the hierarchical model, each input/output port must be associated to the related input/output socket (the port assignment, which maps the port places of the sub-module to the socket places of the substitution transition). The remaining places are internal places, which are only relevant to the SPD functionality module itself; in particular the place *feasible* represents the list of available functionality implementations.

The transitions represent the events, when a transition occurs, it removes tokens from its input places and it adds tokens to its output places. The tokens colours involved in the transition are determined by the arc expressions. This inscription is written in the CPN ML programming language and is built from typed variables, constants, operators, and functions ([2]). The Figure 5-6 and Figure 5-7 show the variables and functions defined in our CPN model.

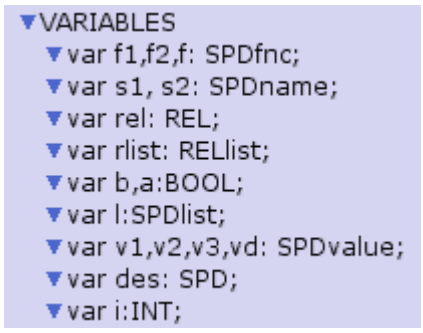

Figure 5-6: the variables

The arc expression are used to define the input-output behaviour, furthermore the arc expressions on the input arcs, together with the tokens on the input places, determine whether the transition is enabled, For an enabled transition it must be possible to find a *binding* of the variables involved in the transition. When a transition occurs with a given binding, i) it consumes, on each input place, the multi-set of token colours, corresponding to the evaluation of the related input arc expression, and analogously ii) it produces on each output place, the multi-set of token colours, corresponding to the evaluation of output arc expression.

```
Functions
fun excl1(r:REL,name1:SPDname,name2:SPDname)=
  if #n1(r)=name1 andalso #n2(r)=name2 then true
  else if #n1(r)=name2 andalso #n2(r)=name1 then true else false;
fun excl(l:RELlist,name1:SPDname,name2:SPDname)=
  if l=[] then false else
  if excl1(hd(l),name1,name2) then true else excl(tl(l),name1,name2)
fun cplck(l:RELlist,name1:SPDname)=
  if l=[] then false else
  if #n1(hd(l))=name1 then true else cplck(tl(l),name1)
fun cpl(l:RELlist,name1:SPDname)=
  if #n1(hd(l))=name1 then #n2(hd(l)) else cpl(tl(l),name1)
fun new(f:SPDfnc,l:SPDlist,e:SPDvalue)=(*add ordinarly*)
  if l=[] then [f] else
  if #v(hd(l))>=e then f::l
  else hd(l)::new(f,tl(l),e)
fun ck(l:SPDlist,e:SPDvalue)=
  if l=[] then false else
  if #v(hd(l))>=e then true
  else ck(tl(l),e);
fun sel(vd:SPDvalue, l:SPDlist)=
  if l=[] then{v=0,n=""} else
  if #v(hd(l))>=vd then hd(l)
  else sel(vd,tl(l));
fun ck1(l:SPDlist,e:SPDvalue,i:SPDvalue)=
  if sel(e,l)={v=0,n=""} then false else (#v(sel(e,l)))<i;
fun not_ckn(l:SPDlist,s:SPDname,vd:SPDvalue)=
  if l=[] then false else
  if #n(hd(l))=s andalso #v(hd(l))<vd then true
  else not_ckn(tl(l),s,vd);
fun ckn(l:SPDlist,s:SPDname,vd:SPDvalue)=
  if l=[] then false else
  if #n(hd(l))=s andalso #v(hd(l))>=vd then true
  else ckn(tl(l),s,vd);
fun seln(s1:SPDname, l:SPDlist)=
  if #n(hd(l))=s1 then hd(l)
  else seln(s1,tl(l));
fun rm a [] = []
  | rm a (x::xs) = if a=x
    then xs
    else x::(rm a xs)
```

Figure 5-7: the functions

In the following there are brief descriptions of relevant transitions in SDP Functionality module:

- *t1* removes the desired value that is expressed as an SPDvalue (integer) from *desired* place and adds it to the place *dsr val*;
- *t2* removes the desired value that is expressed as a SPDname(string) from *desired* place and adds it to the place *dsr name*;
- *select by name*:
    - it removes the desired name (SPDname: string) from the place *dsr name*,
    - it verifies if in the place *feasible* (that represents the available functionality implementations) there is an implementation with name equal to desired name and value greater than or equal to desired value (SPDvalue: integer) taken from place dsr val,
    - if the previous point is satisfied then it adds the candidate functionality implementation to the place *nList*, else no token is added;
- *select by value* adds, to *nList* place, the functionality implementation with the minimum value that is greater than or equal to desired value. Note that this transition is enabled if one or more values associated to the available functionality implementations (one or more record in SDPlist), from place *feasible*, are greater than or equal to token colour on *dsr val* place;
- *change*: this transition change the functionality implementation by removing the token from place *implemented* (the old implementation) and by adding in this place the token removed from place *nList* (new implementation, previously selected);
- *off_fnc* removes an implementation record from feasible place or implemented place; this means that the removed implementation of the functionality has become unavailable.
- *new_fnc* adds a new implementation record to *feasible* place; this means that a new implementation of the functionality is now available.

For the sake of simplicity, we consider that the each functionality can be enabled by three different implementations, each one with its own SPD value, as defined in the follow:

- AUTHENTICATION:

- Password Authentication Protocol (PAP) with SPD value equal to two, thus, its associated record is `{v=2,n="PAP"}`;
- Extensible Authentication Protocol (EAP) with SPD value equal to three, thus, its associated record is `{v=3,n="EAP"}`;
- Challenge-Handshake Authentication Protocol (CHAP) with SPD value equal to eight, thus, its associated record is `{v=8,n="CHAP"}`;

- IDENTIFICATION:
  - PIN with SPD value equal to two, thus, its associated record is `{v=2,n="PIN"}`;
  - Password with SPD value equal to five, thus, its associated record is `{v=5,n="PSW"}`;
  - Token with SPD value equal to eight, thus, its associated record is `{v=8,n="TKN"}`;

As explained previously the initial state of the system is defined by initial marking M_0, in Figure 5-8 - Figure 5-10 are respectively shown the initial marking of the main page system and of the two sub-module identification and authentication.

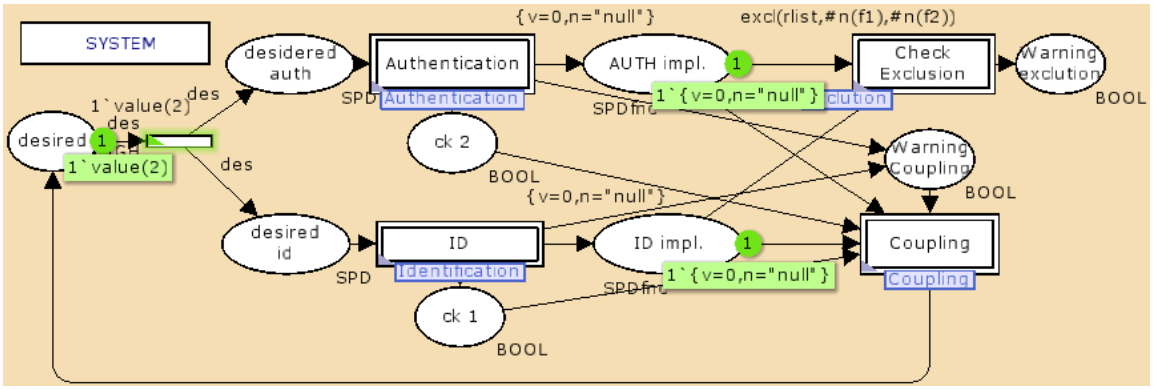

Figure 5-8: Initial Marking of the system

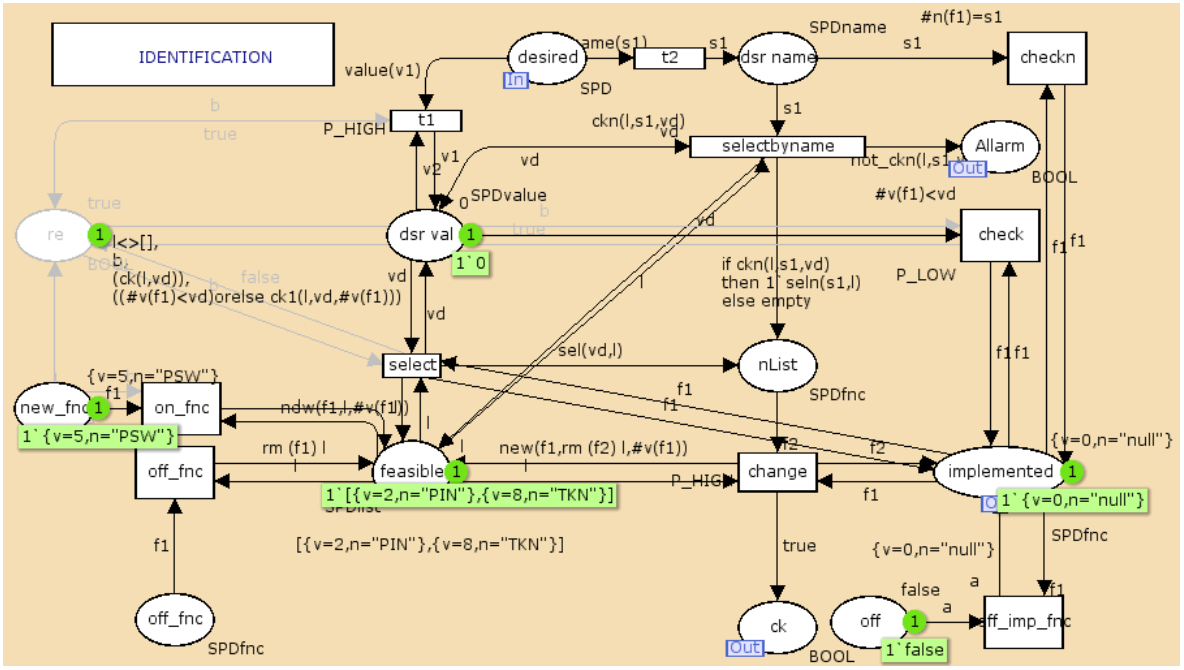

Figure 5-9: initial marking of identification sub-module

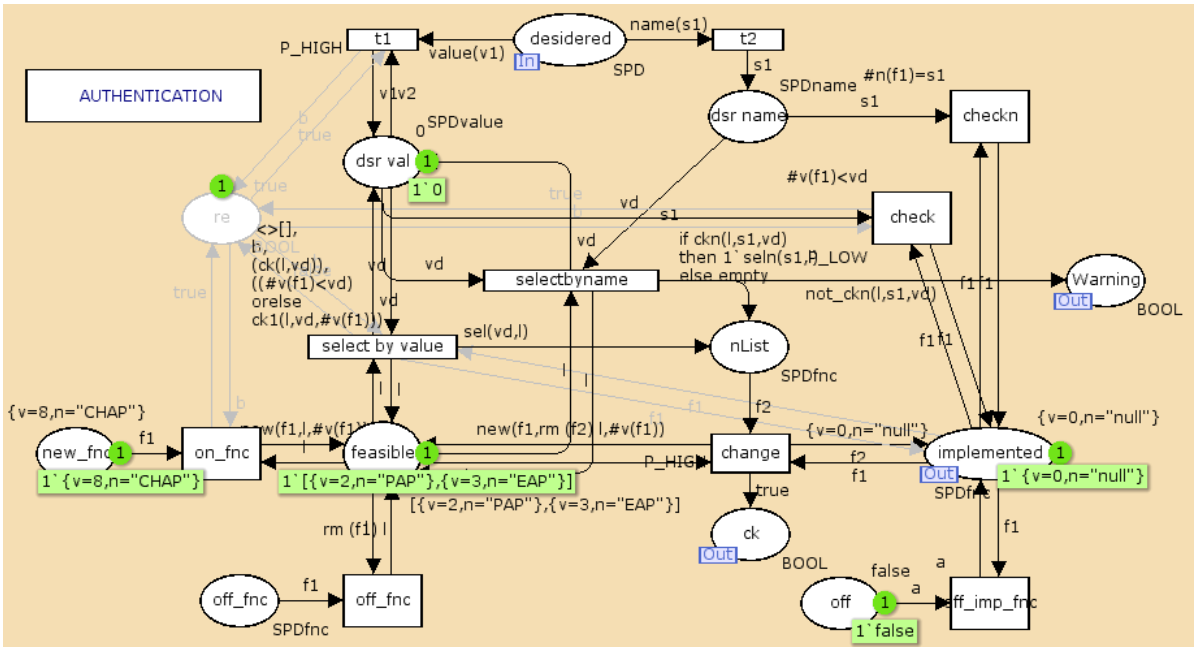

Figure 5-10: initial marking of authentication sub-module

The initial marking of place *desired* contains a single token with colour 2. This means that the variable *des* must be bound to1, and since the arc, coming from *Desired*, is the only input arc of the transition *t,* then this transition is enabled and the only possible binding is: `des = 2`. An occurrence of transition t, in the main page, with this binding, removes the token with colour 2 from the input place *desired* and adds a same token both to *desired auth* and *desired ID*, according to the result of evaluating the arc expression. Figure 5-11 shows the CPN model in the new marking M_1.

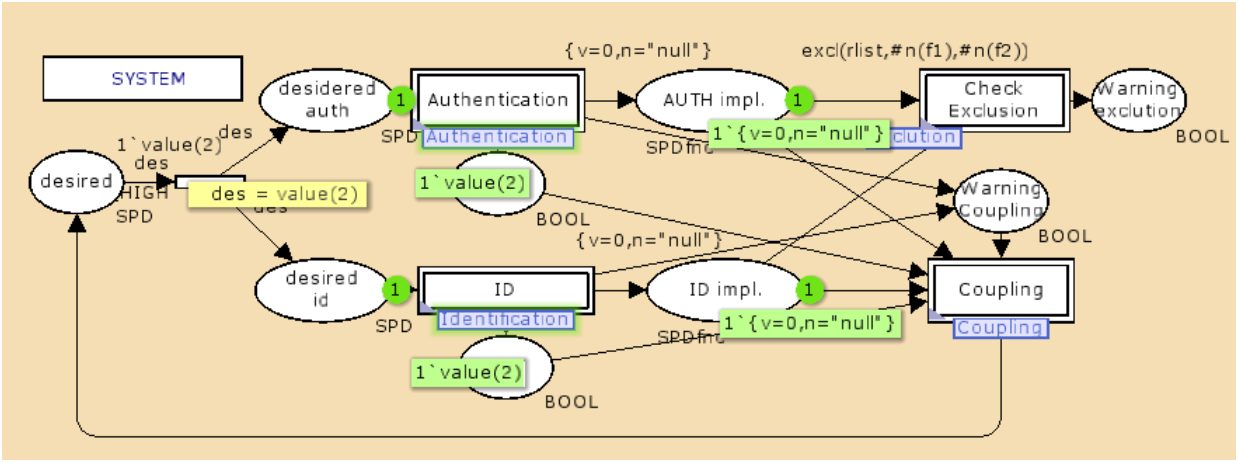

Figure 5-11: Marking M_1 reached when t occurs in M_0

Considering the marking M_1 all the substitution transitions are concurrently enabled. The fragment of interest in each sub-module is shown in Figure 5-12. In other words the place *desired*, in all subpages, contains the token colour 2, thus the transition t1 are concurrently enabled in all sub-modules, with only possible following binding: `v1 = 2, v2 = 0`. Note that the transition t2 is disabled because the token colour in the place *desired* is not a string (SPDname colorset) but an integer (SPDvalue colorset).

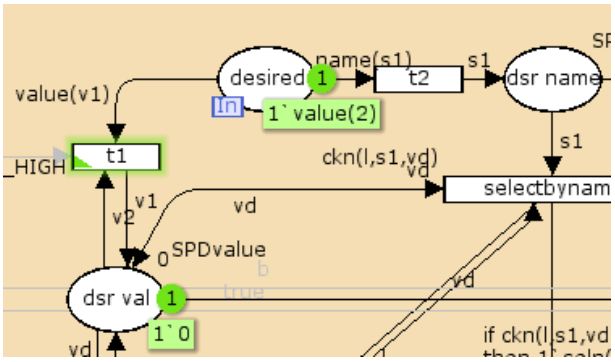

Figure 5-12: Marking M_1 in the sub-modules

Considering, in the sub-module authentication, the occurrence of transition t1, with the binding previously defined, then the new marking M_2 is shown in Figure 5-13.

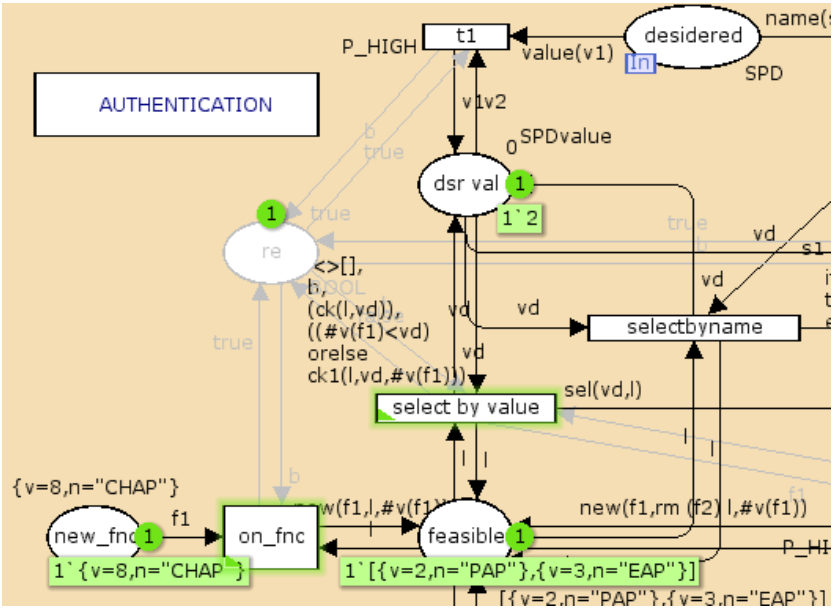

Figure 5-13: Marking m_2 in the authentication sub-module

As shown in Figure 5-13, the transition *select by value* is enabled, in particular the expressions on the arcs of the input and output are evaluated and the guard conditions (the inscriptions at the top right of the transition) are met. In the follow, the guards are detailed:

- `l<>[]`: in the place *feasible* there is at least one implementation available,
- `b` is a Boolean value used to enable the transition only after a significant change, such as the desired value,
- `(ck(l,vd))` this function returns true if in the place *feasible* there is an implementation with the associated SPD value greater than or equal to the desired one. In other words there is an available implementation that satisfies the requirement on SPD value:

```
fun ck(l:SPDlist,e:SPDvalue)=
if l=[] then false
```

```
else if #v(hd(l))>=e then true
else ck(tl(l),e),
```

- `((#v(f1)<vd)orelse ck1(l,vd,#v(f1)))` this condition is true if the SPD value associated to the implemented functionality (in place *implemented*) is less than desired value or in place *feasible* is available an implementation with the associated SPD value that is i) greater than or equal to the desired one and ii) less than the value of implemented functionality.

Suppose that an event occurs and the transition *select by value* fires. The Figure 5-14 shows the binding (the yellow box) and new marking M_3 in the AUTH sub-module. It is important to note that the output value is not shown; in fact this value is evaluated by the following function:

```
fun sel(vd:SPDvalue, l:SPDlist)=
if l=[] then{v=0,n=""} else
if #v(hd(l))>=vd then hd(l)
else sel(vd,tl(l));
```

In particular this function returns the record with the minimum value that is greater than or equal to the desired one.

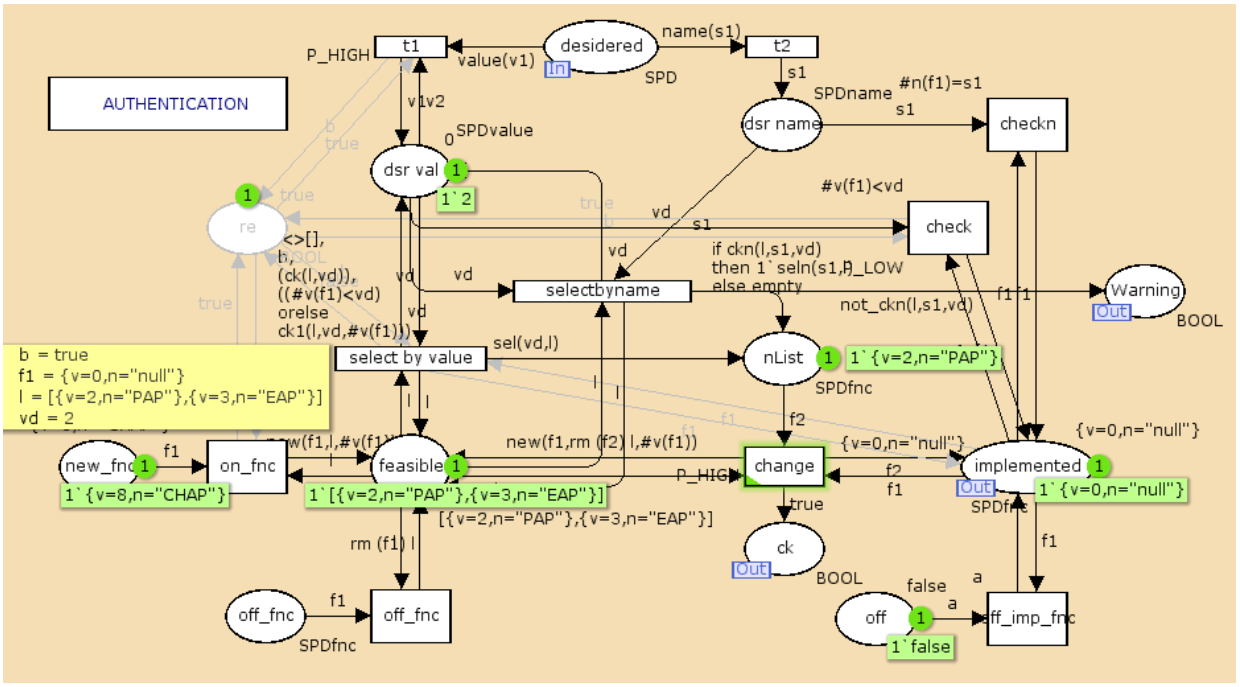

Figure 5-14: Marking m_3 in the authentication sub-module

The Figure 5-15 shows the system state (marking M_f) after all transitions in both sub-modules are fired, in particular, the transitions are *change* and *on func* in the sub-module AUTH, and *t1*, *select by value*, *change* and *on func*, in the sub-module ID.

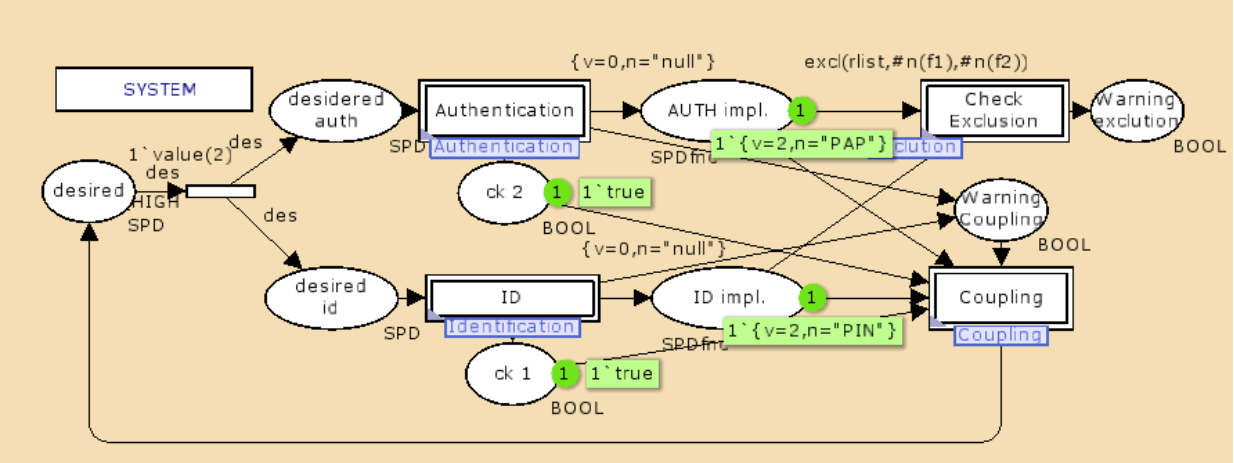

Figure 5-15: the marking M_f in the system page

The Coupling Relation sub-module (Figure 5-16), corresponding to the substitution transition *Coupling* in the main page shown in Figure 5-8. In this sub-module we have three input port: the place *warning*, the place *ck* and the place *ck1*; an output port, the place *desired*. The relevant internal places are *ID* and *AUTH*; these places represent the list of the functionalities that requires a particular coupling. For example, as shown in Figure 5-16 (initial marking of ID) we suppose that the PIN implementation of ID functionality requires the EAP implementation. In particular, the Figure 5-16 shows the marking M_f in the Coupling Relation sub-module; it is important to note that the transition *Coupling ID* is enabled, meaning that there is a coupling constraint to satisfy.

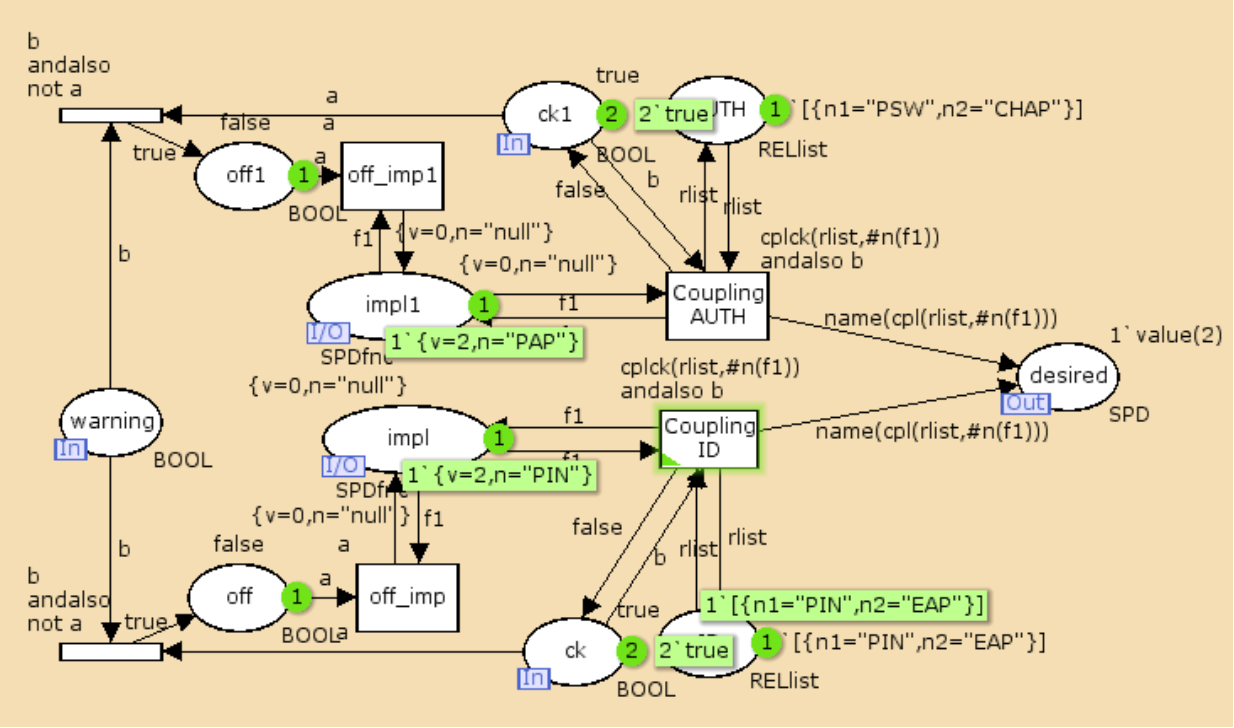

Figure 5-16: The Coupling relation sub-module

The following Figures (from Figure 5-17 to Figure 5-20) show how the system model "detects" a coupling constraint and then "reacts" to satisfy it. In particular, the marking M_{f1}, obtained after that the *Coupling ID* transition fires, is shown in the Figure 5-17 and in the Figure 5-18, respectively, considering the Coupling Relation Sub-module and the System page. These figures, simply, show that a token, that carries the name of the implementation to be enabled in order to satisfy the coupling constraint, is added to place *desired*. The Figure 5-19 shows an intermediate marking M_{fn} in the Authentication sub-module, after the firing of transition *selecy by name*, to highlight that the transition *change* is enabled and the place *Warning* contains a token carrying the value `false`; meaning that both constraint (desired SPD value and coupling constraint) are met. Finally the Figure 5-20 show the final marking M_{ff} in the system page.

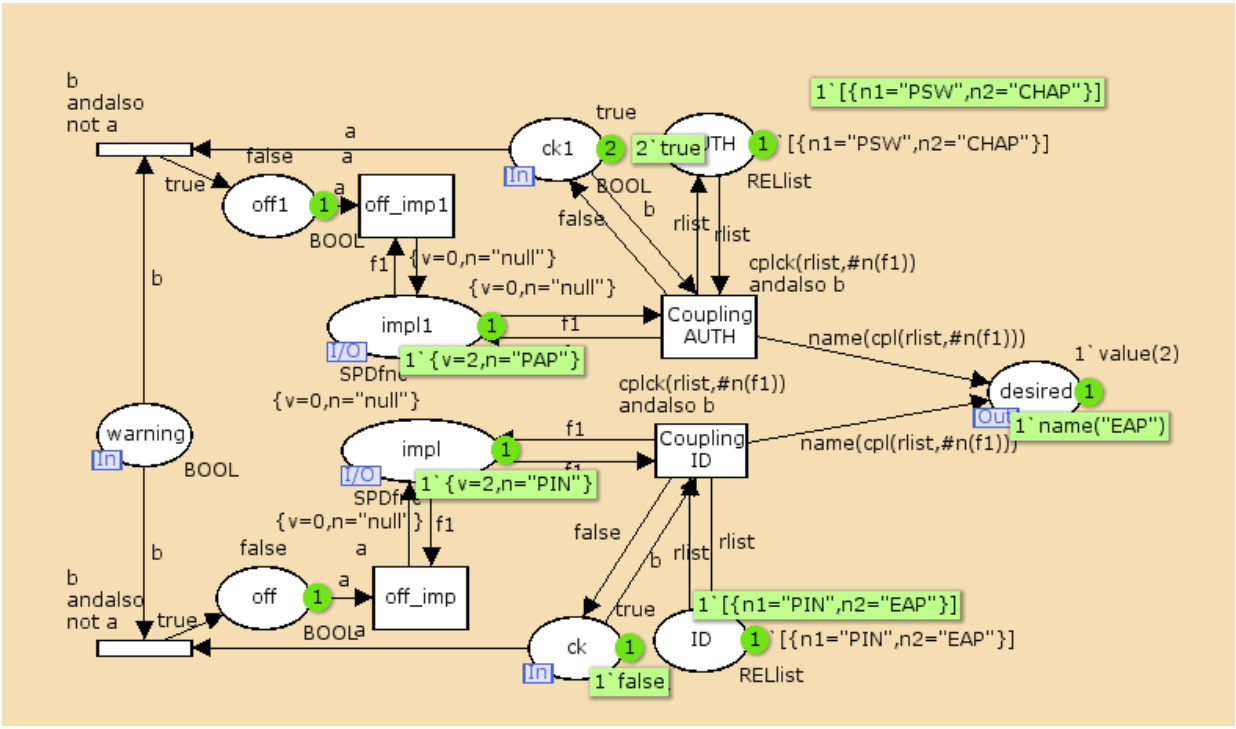

Figure 5-17: the marking M_{f1} in the coupling relation sub-module

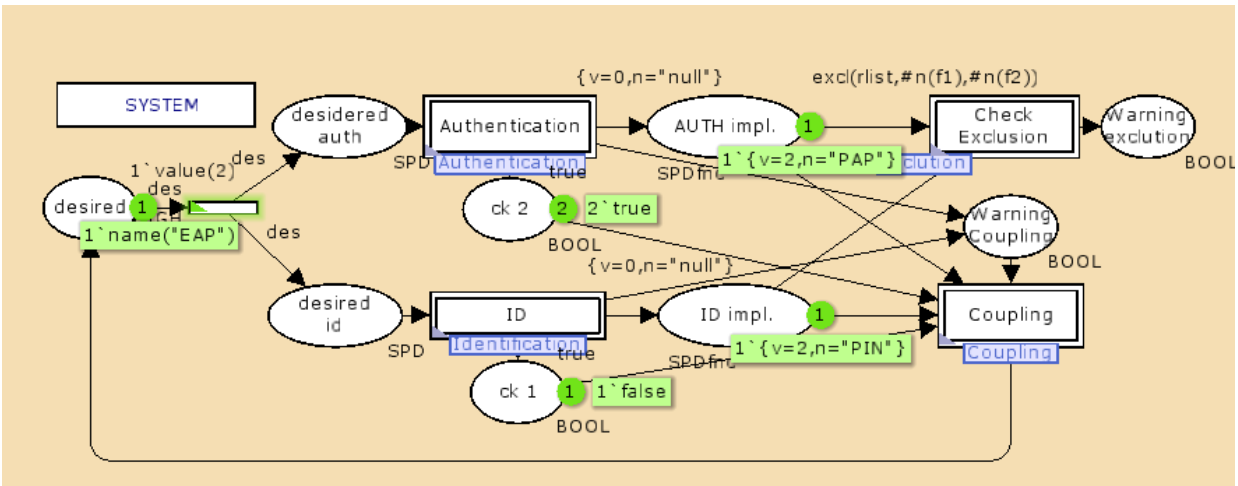

Figure 5-18: The Marking M_{F1} IN THE system Page

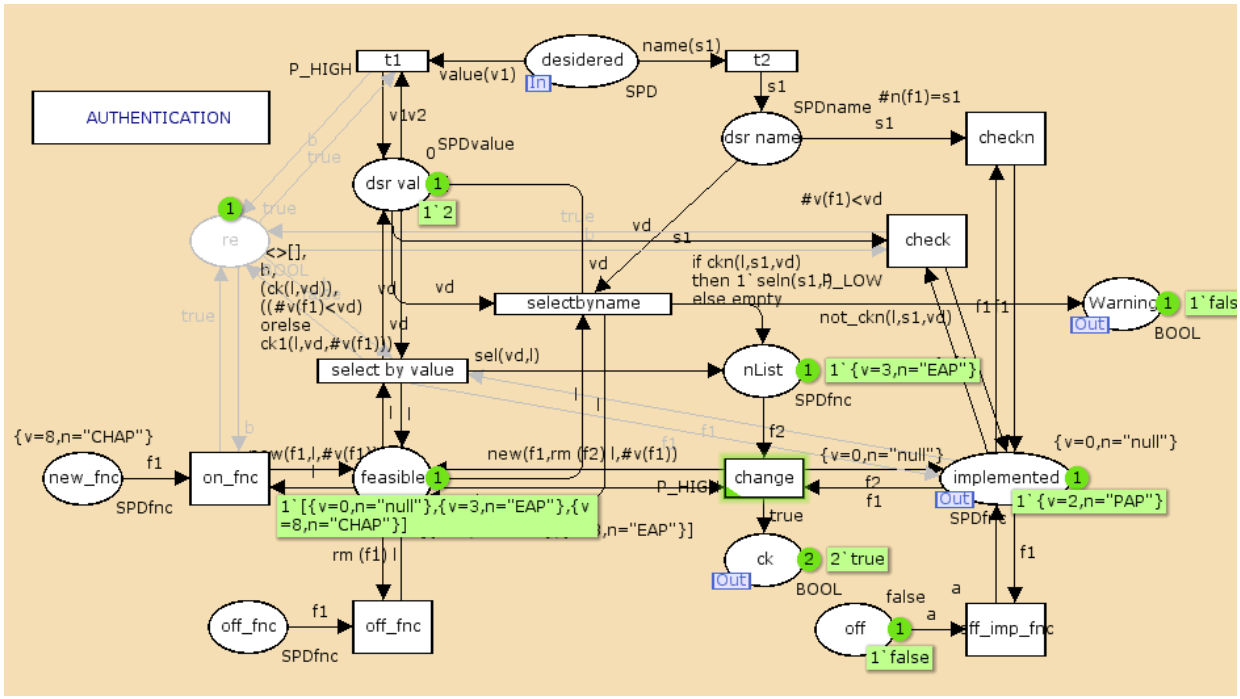

Figure 5-19: The marking M_{Fn} in the AUTHentication SUB-MODULE

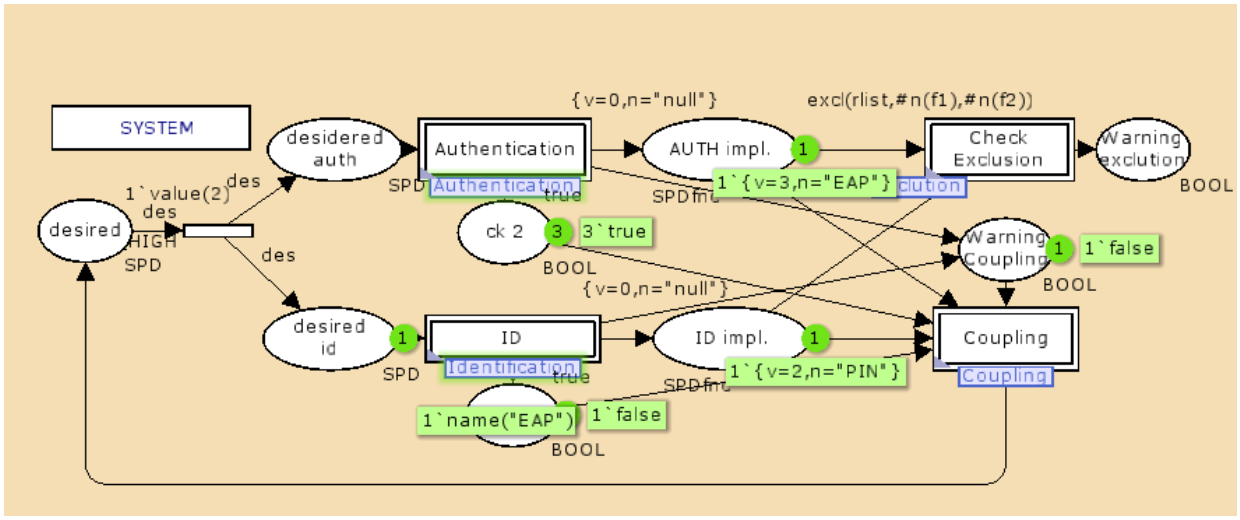

Figure 5-20: The marking M_{Ff} in the System page

6 Conclusions

In this document the major prototypes for the SHIELD Middleware have been included, with the objective of providing the building blocks for the common platform and the demonstrator.

The major achievements are:

- the intrusion detection bundle, that implement the monitoring and filtering capabilities required for a “real time” control of SPD level,
- the definition of the Middleware protection profile, i.e. the first step towards the SHIELD standardization
- the identification of a new semantic to describe the SHIELD system
- the identification and instantiation of a framework for Policy Based Management/Access.

Further results will be developed in the second phase of the project, trying to:

i. enrich the current solutions,

ii. define new ones and, above all,

iii. harmonize the components with the metric approach

The final target is to build a common Middleware platform that, using these components, is really able to implement the SPD composability.

Advances with respect to this document will be available in D5.4.

7 References

[1] A.V. Ratzer, L. Wells, H.M. Lassen, M. Laursen, J.F. Qvortrup, M.S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. *Proc. of 24th International Conference on Applications and Theory of Petri Nets (Petri Nets 2003). Lecture Notes in Computer Science 2679*, pp. 450-462, Springer-Verlag Berlin, 2003.

[2] K. Jensen, L.M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer (STTT)* 9(3-4), pp. 213-254, 2007.