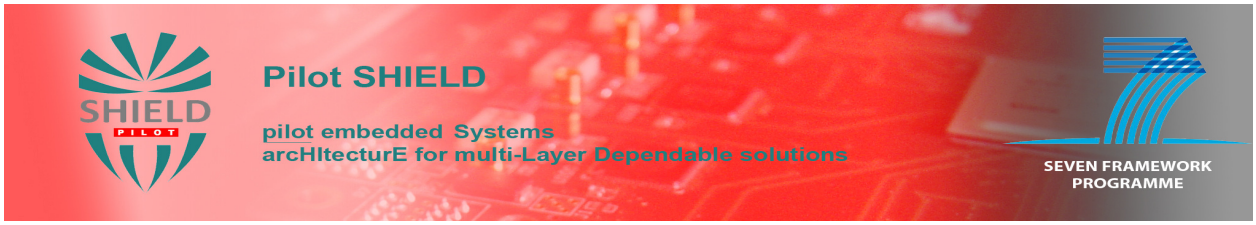

Project no: 100204

p-SHIELD

pilot embedded Systems architecture for multi-Layer Dependable solutions

Instrument type: Capability Project

Priority name: Embedded Systems (including RAILWAYS)

D5.2: SPD middleware and overlay functionalities report

Due date of deliverable: M18 (30th September 2011)
Actual submission date: M19 (15th November 2011)

Start date of project: 1st June 2010 Duration: 19 months

Organisation name of lead contractor for this deliverable: SE

Revision [Version 1.0]

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 Fiaschetti	Univ. “La Sapienza”		
Andrea Morgagni	Elsag Datamat		
Renato Baldelli	Elsag Datamat		
Jose verissimo	CS		
Mohammad Chowdhury	CWIN		
Andrea Taglialatela	TRS		
Vincenzo Suraci	Univ. “La Sapienza”		
Andi Palo	Univ. “La Sapienza”		
Gordana Mijic	THYIA		
Spase Drakul	THYIA		
Ljiljana Mijic	THYIA		
Nastja Kuzmin	THYIA		
Mohammad Chowdhury	CWIN		
Carlo Bruni	Univ. “La Sapienza”		
Roberto Cusani	Univ. “La Sapienza”		
Gaetano Scarano	Univ. “La Sapienza”		
Francesco Delli Priscoli	Univ. “La Sapienza”		
Alberto Isidori	Univ. “La Sapienza”		
Giorgio Koch	Univ. “La Sapienza”		
Claudio De Persis	Univ. “La Sapienza”		
Antonio Pietrabissa	Univ. “La Sapienza”		
Reviewed by			
Name	**Company**		
Josef Noll	Movation		
Sarfraz Alam	CWIN		
Andrea Fiaschetti	Univ. “La Sapienza”		
Approved by			
Name	**Company**		
Andrea Morgagni	Elsag Datamat		

Modification History		
Issue	**Date (DD/MM//YY)**	**Description**
Version 1	15/11/2011	First version

Contents

Figures

Tables

Acronyms

Acronym	Meaning
ESD	Embedded System Device
ESs	Embedded Systems
L-ESD	Legacy Embedded System Device
MS	Middleware Service
MwA	Middleware Adapter
NC	Node Capability
NoA	Node Adapter
NS	Network Service
NwA	Network Adapter
pS-ESD	pSHIELD Embedded System Device
pS-MS	pSHIELD Middleware Service
pS-OS	pSHIELD Overlay Service
pS-P	pSHIELD Subsystem
pS-P	pSHIELD Proxy
pS-SPD-ESD	SPD Embedded System Device
SPD	Security Privacy Dependability
CC	Common Criteria
FUA	Faults with Unauthorized Access
HMF	Human-Made Faults
NFUA	Not Faults with Unauthorized Access
NHMF	Nonhuman-Made Faults
SoC	System on Chip

- This page intentionally left blank -

1 Executive Summary

The purpose of this document is to report the results obtained by the pSHIELD project in WP5 with respect to:

- the pSHIELD Middleware and its core SPD services,
- the Policy Based Management
- the Overlay architecture and control algorithms.

Particular attention has been devoted to the most innovative aspect: the composability functionality performed by the Middleware and Overlay, that is the key enabling technology for the pSHIELD framework. This concept is depicted in Figure 1 as a closed loop problem and the WP5 activities covered by this deliverable and necessary to enable it are highlighted (Tasks 5.2, 5.3 and 5.4 as described in the Technical Annex).

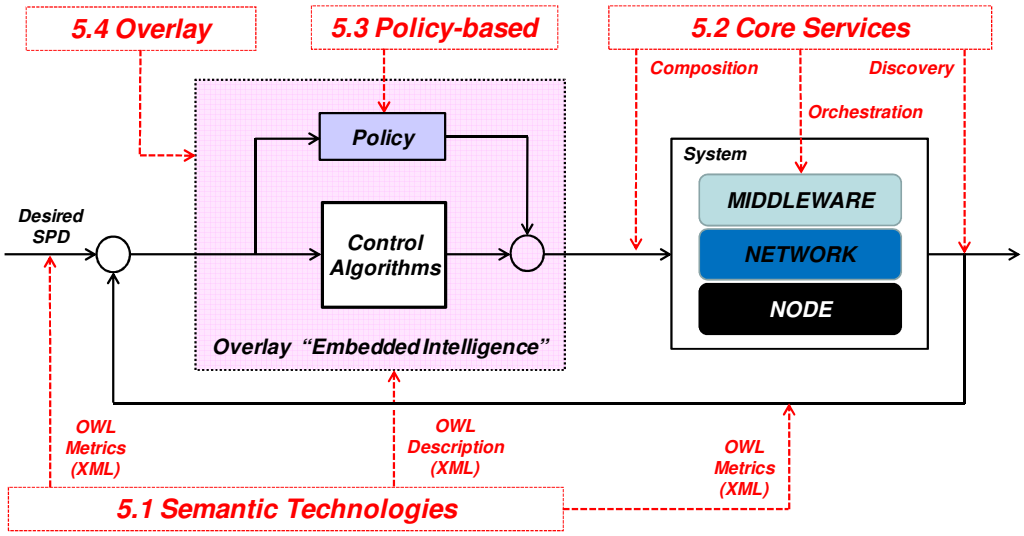

Figure 1 WP5 Composability Concept

Since pSHIELD is a pilot project, the solutions proposed in this document don't pretend to be exhaustive and optimal, but their purpose is to realize a proof of concept of the pSHIELD key concepts in a reduced (but significant) scope and in a simple scenario, putting the bases for further improvements and investigations.

The background analysis, state of the art and performance considerations carried out in this report shall be considerent in conjunction with the prototypes already presented in deliverable D5.2. These documents together cover all the work carried out by WP5 partners.

The document is structured as follows: after the punctualization of terms and definitions in Section 2 and the contextualization of the achieved results in Section 3, the Section 4 is focused on the pSHIELD Middleware and core SPD services (based on the OSGi framework)., enriched by cross reference with semantics in Section 5. In Section 6 the rationale and architectural analysis of potential implementation of Policy Based Management in pSHIELD environment is reported. Last, but not least, in Section 7 the Overlay architecture and behaviour is described, with particular attention to the underlying control theory

2 Terms and definitions

This section lists the applicable documents

Ref	Document Title	Issue/Date
TA	pSHIELD Technical Annex	1
M0.1	Formalized Conceptual Models of the Key pSHIELD Concepts	1
M0.2	Proposal for the aggregation of SPD metrics during composition	1
D2.1.1	pSHIELD Systems requirements and specifications	1
D2.2.1	pSHIELD metrics definition	1
D5.1	pSHIELD Semantic Models	1
D5.2	pSHIELD SPD Middleware and Overlay Functionalitity prototypes	1
D5.3	pSHIELD Semantic Models report	1

2.1 SPD Dictionary

A comprehensive dictionary of the SPD concepts is provided by the project glossary included in Annex 1.

3 pSHIELD Middleware & Overlay contextualization

Going a little more in detail with the well known closed-loop representation depicted in the executive summary, the main tangible result of WP5 is the formulation (and implementation) of the SPD composability mechanism, enriched with common criteria rationale and context aware optimization.

The pSHIELD Middleware is the logical and software entity in charge to realize such behavior, as well as to implement basic (atomic) security services. As already presented in D5.2, different prototypes have been delivered to demonstrate the effectiveness of the middleware technologies and algorithms to address the SPD-composition problem.

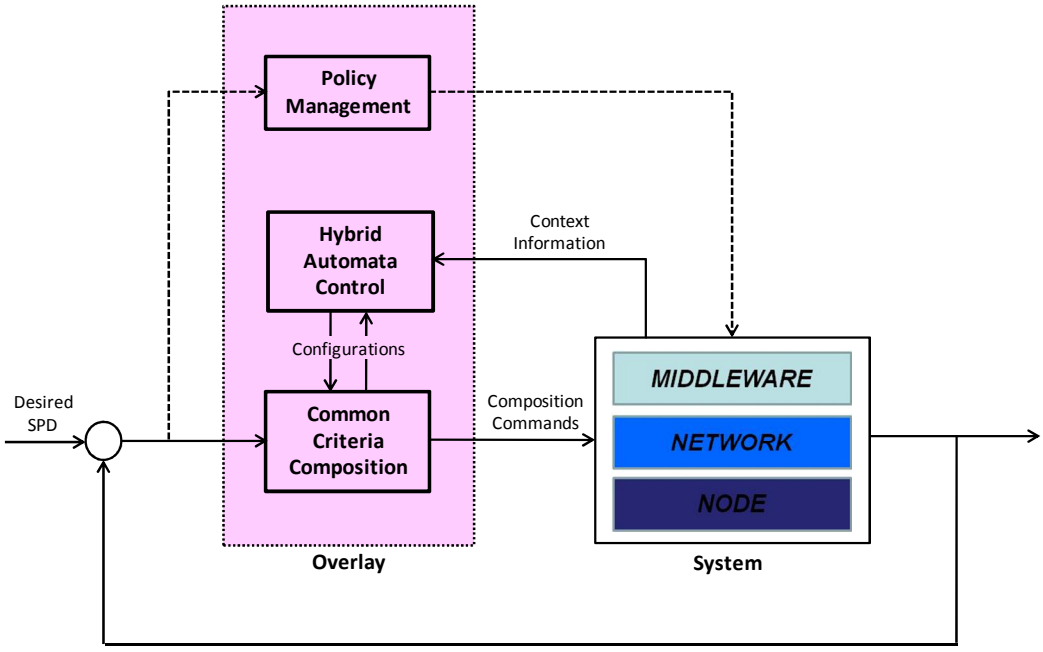

Figure 2 pSHIELD Middleware detailed rationale

In Figure 2 the overall rationale at the basis of pSHIELD middleware is depicted: thanks to the Middleware core services and the semantic representation, the system's elements can be discovered by the overlay (a logical entity embedded in the middleware itself); then, the overlay is able to compose them to achieve the desired SPD level in two separate ways: i) by adopting policies or ii) by following the Common Criteria composition approach (defined in WP2) mixed to the context-aware Hybrid Automata approach.

The different blocks have been separately designed and developed by means of individual prototypes and approaches (simulations for some components, architectural design for some others, software implementation for others). In particular the following prototypes have been obtained:

- **OSGI framework** to perform Middleware Core Services for discovery and composition of pSHIELD components

- **OWL file** representing the pSHIELD ontology that, together with the pSHIELD middleware, makes the composition possible. In particular this prototype includes the **reasoner** for Common Criteria compliant composition of SPD metrics.

- **Architectural design** and performances analysis of a Policy Based approach by which the middleware composition could be driven

- Matlab **simulation** and theoretical **formalization** of an Hybrid Automata approach to drive the SPD composition in a context-aware way

The integration of a subset of this elements has been done in the final pSHIELD demonstrator.

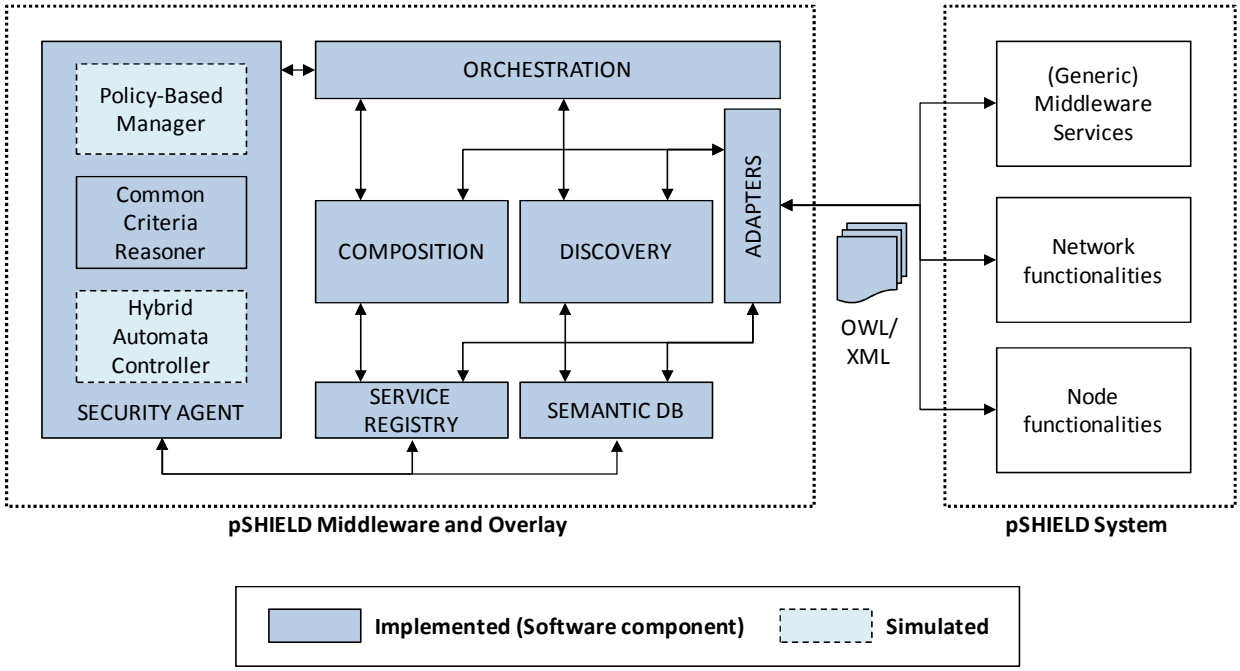

Figure 3 Middleware Prototypes Integration

In Figure 3 the rationale of Figure 2 is translated into the prototype output: the Core SPD Services (OSGI Framework), the pSHIELD Ontology (OWL) and the Common Criteria Reasoner have been implemented in a Java software enviroment, while the policy-based management and the hybrid automata controller have been designed and simulated in other context.

4 pSHIELD Middleware and core SPD Services

4.1 From concept to architecture

The core SPD services are a set of mandatory basic SPD functionalities provided by a pSHIELD Middleware Adapter in terms of pSHIELD enabling middleware services. The core SPD services aim to provide a SPD middleware environment to actuate the decisions taken by the pSHIELD Overlay and to monitor the Node, Network and Middleware SPD functionalities of the Embedded System Devices under the pSHIELD Middleware Adapter control. The following core SPD services are provided:

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

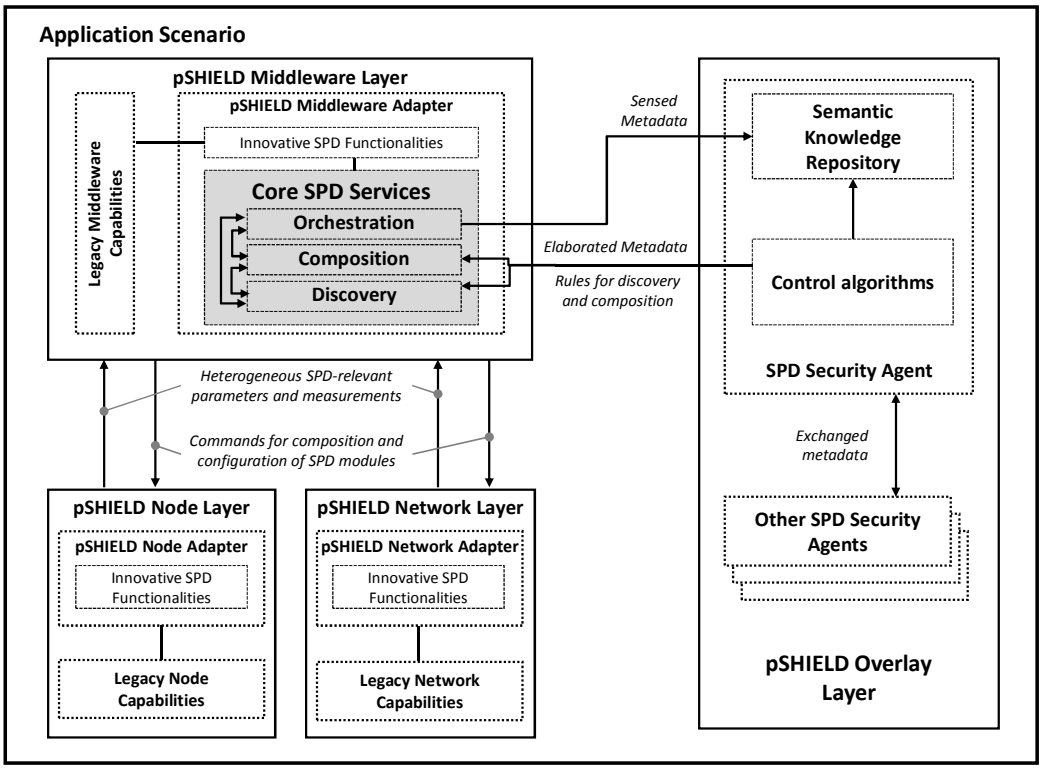

Figure 4 Core SPD services in the pSHIELD functional component architecture

Service discovery allows any pSHIELD Middleware Adapter to discover the available SPD functionalities and services over heterogeneous environment, networks and technologies that are achievable by the pSHIELD Embedded System Device (pS-ESD) where it is running. Indeed the pSHIELD secure service discovery uses a variety of discovery protocols (such as SLP[1], SSDP[2], NDP[3], DNS[4], SDP[5], UDDI[6]) to harvest over the interconnected Embedded System Devices (ESDs) all the available SPD services, functionalities, resources and information that can be composed to improve the SPD level of the whole system. In order to properly work, a discovery process must tackle also a secure and dependable service registration, service description and service filtering. The service registration consist in advertising in a secure and trusted manner the available SPD services. The advertisement of each service is represented by its formal description and it is known in literature as service description. The registered services are

[1] IETF Service Location Protocol V2 - http://www.ietf.org/rfc/rfc2608.txt
[2] UPnP Simple Service Discovery Protocol - http://upnp.org/sdcps-and-certification/standards/
[3] IETF Neighbour Discovery Protocol - http://tools.ietf.org/html/rfc4861
[4] IETF Domain Name Specification - http://www.ietf.org/rfc/rfc1035.txt
[5] Bluetooth Service Discovery Protocol
[6] OASIS Universal Description Discovery and Integration - http://www.uddi.org/pubs/uddi_v3.htm

discovered whenever their description matches with the query associated to the discovery process, the matching process is also known in literature as service filtering. On the light of the above a SPD services discovery framework is needed as a core SPD functionality of a pSHIELD Middleware Adapter. Once the available SPD services have been discovered, they must be prepared to be executed, assuring that the dependencies and all the services preconditions are validated. In order to manage this phase, a service composition process is needed.

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

Service orchestration is in charge to deploy, execute and continuously monitor those SPD services which have been discovered and composed. This is part of the pSHIELD Middleware Adapter functionality. While service composition works "off-line" triggered by an event or by the pSHIELD Overlay, service orchestration works "on-line" and is continuously operating in background to monitor the SPD status of the running services.
The Orchestration, Composition and Discovery functionalities are the enablers (i.e. the sensors and the actuators) of the decisions taken by the pSHIELD Security Agent Control Algorithms residing in the pSHIELD Overlay. The mutual interoperation between the pSHIELD Middleware Adapter and the pSHIELD Security Agent enables the pSHIELD Composability concept.
It is worth to note that not all the core SPD services must be necessarily located in each pSHIELD Embedded System Device (pS-ESD). Indeed the pSHIELD component architecture depicted in Figure 4 identifies the Discovery, Composition and Orchestration functionalities that must be supported by at least one pS-ESD in a network of Embedded System Devices. Moreover the core SPD services can be deployed applying centralized or distributed approaches. It is a matter of the precise application scenario to decide whether a specific functionality must be supported by each Embedded System Device (ESD). It is obvious that the more ESDs are equipped with the pSHIELD Middleware Adapter (resulting to be a pS-ESD), the more will be the coverage area and the effectiveness of the pSHIELD functionalities to guarantee a certifiable SPD level (based on common shared SPD metrics) over the whole system.

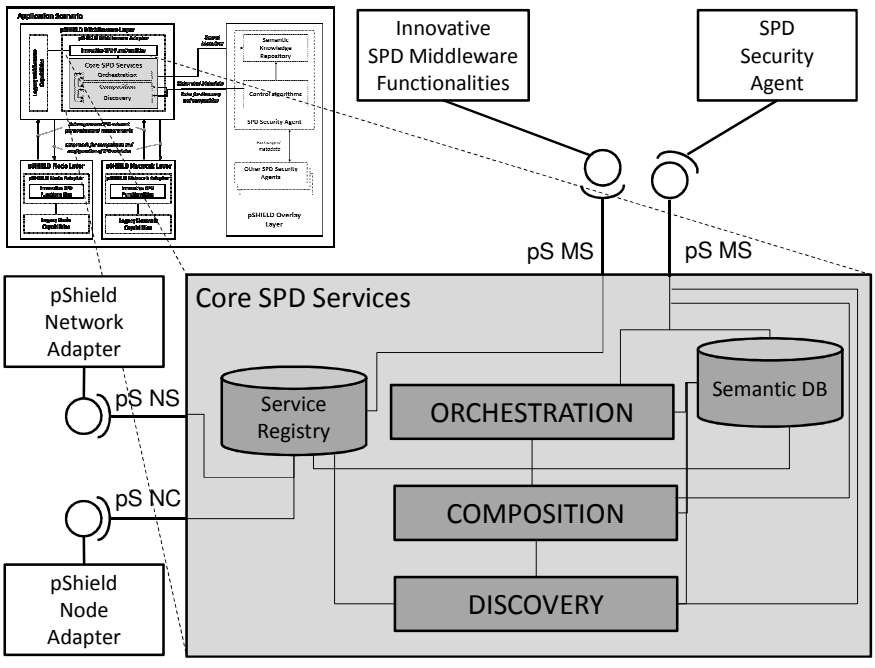

Figure 5 Core SPD services conceptual framework

Let see more in detail the formalized conceptual model of the Core SPD services, detailing the architecture depicted in Figure 4 and exploding the core SPD services into their functional components, in compliance with the pSHIELD functional architecture described in deliverable D2.3.1.

Apart the Discovery, Composition and Orchestration components already described in the previous section, the following additional conceptual entities have been introduced:

- **Service Registry**: it acts as a database to store the service entries (e.g. the SPD components description of provided functionalities, interfaces, semantic references, etc.). Any pSHIELD Node, Network or Middleware layer component can be registered here to be discovered.
- **Semantic Database**: it holds any semantic information related to the pSHIELD components (interface, contract, SPD status, context, etc.). The use of common SPD metrics and of a shared ontology to describe the different SPD aspects involved in guaranteeing a precise level of SPD, allows to dominate the intrinsic heterogeneity of the SPD components. Any semantic data is thus technology neutral and it is used to interface with the technology independent mechanisms applied by the pSHIELD Overlay.

Focusing exclusively on the Core SPD services located in the pSHIELD Middleware Adapter, we can describe how it works when it is in an operative status. Let consider a typical situation, where the whole system is properly working at runtime. The Orchestration functionality is in charge to monitor continuously the Semantic DB with the updated status of the functionalities operating at node, network and middleware layers. The pSHIELD Adapters are in charge to update in the Semantic DB (by proper Semantic hubs that, for the sake of simplicity, have not been shown in Figure 5) their status.
Whenever the needed application SPD level, for any reason, due to external/internal unforeseen/ predictable events, changes and goes beyond the threshold, the Orchestrator triggers the Overlay. The Overlay try to react and to restore the SPD level back to an acceptable level identifying the best configuration rules. The Discovery and Composition are then triggered by the pSHIELD Overlay with the aim to apply the configuration rules. On the basis of the configuration rules, the Composition service make use of the Discovery service to search for all the needed and available SPD components. In particular the Composition uses the Discovery mechanism to look for the available SPD component interfaces and contracts over the network. Then the Composition, on the basis of the configuration rules provided by the Overlay, determines which SPD components are required, which should be activated and in which order to make the configuration of SPD components properly work. Thus while the Overlay operate in a technology independent fashion, the Composition service operates all the needed low-level, technology-dependent activities to actuate the Overlay decisions.
From the above description, it is clear that a key role is played by the Discovery mechanism. In pSHIELD the main focus of the Discovery is to look for any available, composable SPD component over the network. In order to cope with this important functionality the Discovery core SPD service must be decoupled into several elementary elements each deputed to a proper functionality.

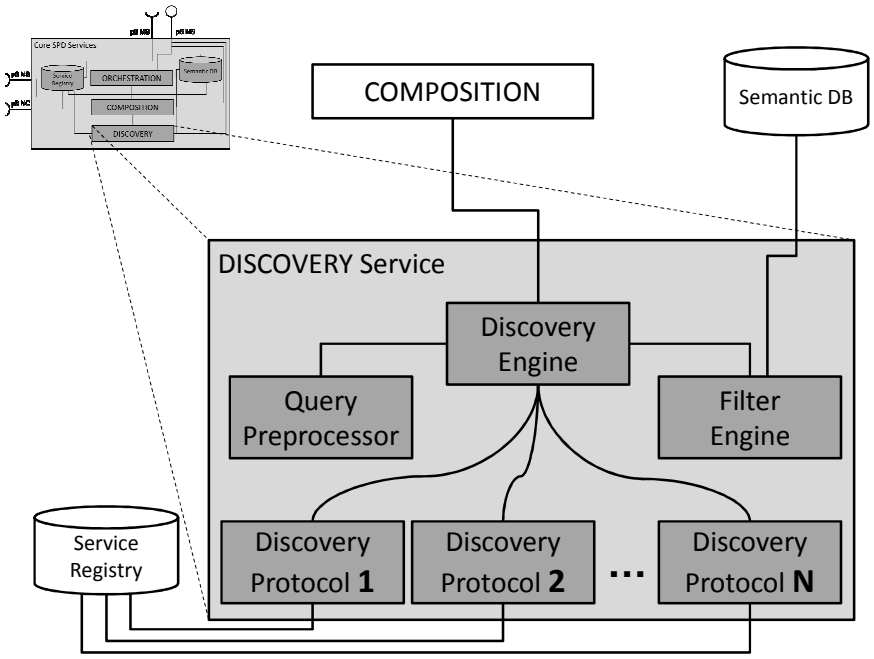

Figure 6 Details of the Discovery core SPD service

Zooming more in the detail the Discovery service, as shown in Figure 6, the following elements can be distinguished:

- **Discovery Engine**: it is in charge to handle the queries to search for available pSHIELD components sent by the Composition service. The Discovery Engine manages the whole discovery process and activates the different functionalities of the Discovery service: (i) the query pre-processor to enrich semantically and contextually the query, (ii) the different discovery protocols to harvest over the interconnected systems all the available SPD components, (iii) the Filter Engine to discard those components not matching with the enriched query;
- **Query Pre-processor:** it is in charge to enrich the query sent by the Composition service with semantic information related to the peculiar context. The query pre-processor can be configured by the Overlay to take care of the current environmental situation;
- **Discovery Protocol:** it is in charge to securely discover all the available SPD components description stored in the Service Registry, using a specific protocol (e.g. Service Location Protocol – SLP or Universal Plug and Play Simple Service Discovery Protocol – UPnP SSDP, etc.). Indeed the SPD component descriptions can be registered in different types of Service Registries, located everywhere in the network, using heterogeneous protocols to be inquired;
- **Filter Engine:** it is in charge to semantically match the query with the descriptions of the discovered SPD components. In order to perform the semantic filtering, the Filter Engine can retrieve from the Semantic DB the information associated to the SPD components, whose location is reported in the description of the SPD component.

The Composition engine tries to accomplish the pSHIELD Overlay configuration rules applying the following procedure:

1. Composition service triggers the Discovery service, sending a SPD component request, looking for those SPD components defined in the configuration rules provided by the Overlay;
2. The Discovery Engine sends the request to the Query Preprocessor;
3. The Query Preprocessor enriches the service request with contextual information and sends it back to the Discovery Engine;
4. The Discovery Engine applies a global service discovery using heterogeneous Discovery Protocols, in order to collect as much available SPD functionalities as possible over the networked Embedded System Devices;
5. Each Discovery Protocol interacts with the Service Registries reachable in the network and retrieves the SPD components' descriptions and provides them back to the Discovery Engine;
6. The Discovery Engine collects the discovered descriptions and sends them to the Filter Engine;
7. The Filter Engine applies a semantic filtering, retrieving the semantic metadata from the semantic DB, accordingly with the references reported in each SPD component description. The filtered list of component is then sent back to the Discovery Engine;
8. The Discovery Engine sends the list of available, filtered SPD components to the Composition service;
9. If the Composition service, considering the available SPD components is able to provide a new configuration, these components are activated, otherwise the Composition service advise the Overlay that it is not possible to apply its decision.

It is important to note that the validity of this conceptual framework model is independent from the specific application scenario. On the basis of this conceptual framework it is possible to derive a number of possible alternative implementations, belonging to different pSHIELD compliant technology providers. If the interfaces and the operation between the different elements are respected, it is possible to setup heterogeneous systems with the enhanced pSHIELD SPD functionalities.
However being pSHIELD a pilot project, a targeted demonstrator will be described in the following. Starting from the conceptual framework depicted above, an instance of the presented framework will be derived in order to achieve the target SPD objectives defined in the pSHIELD application scenario.

4.2 The Innovative SPD Functionalities

The core SPD services introduced in the previous section are in charge to discover, compose and orchestrate those Innovative SPD Functionalities provided by any specific application scenario. An Innovative SPD Functionality can be developed from scratch or can be developed starting from an already existing legacy SPD functionality. In order to make any legacy SPD functionality to be an Innovative SPD Functionality, it must be discoverable, composable and orchestrable.
The legacy device components, i.e. the SPD functionalities already present in the legacy devices, can be classified in Node, Network and Middleware legacy device components according to whether they have been included in a legacy Node, Network or Middleware layer.

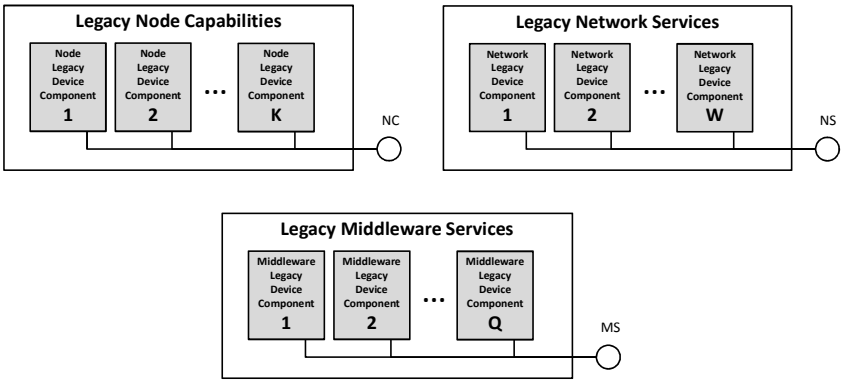

Figure 7 Legacy device components

The legacy functionalities interact with the surrounding world through proprietary interfaces (namely the NC, NS and MS interfaces). To cope with this heterogeneity, it is necessary to introduce an intermediate component between the legacy world and the pSHIELD framework. The role of mediator is played by the pSHIELD Adapter.
The pSHIELD Adapters hosts those Innovative SPD Functionalities that have been ad hoc developed for the pSHIELD project to let the legacy functionalities to be correctly used by the pSHIELD framework. The pSHIELD Adapter allows the legacy functionalities to be discovered, composed and orchestrated. The pSHIELD Adapters can be classified in Node, Network and Middleware pSHIELD-specific components according to whether they are included in the pSHIELD Node, Network or Middleware layer. The Adapters can be directly accessed by pSHIELD Middleware Core SPD Services through the pS-NC, pS-NS and pS-MS interfaces.

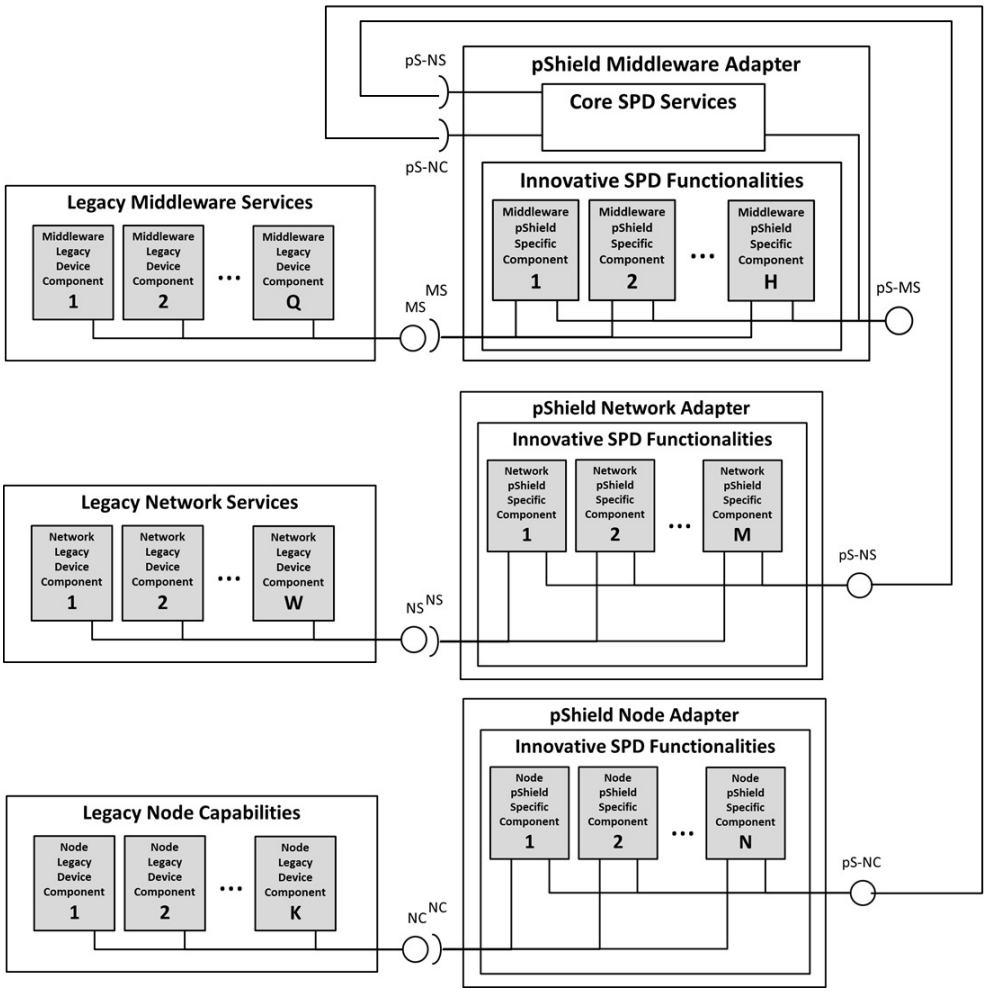

Figure 8 pSHIELD Components' interactions

Any Innovative SPD Functionality must be discoverable and composable. To be discoverable, an Innovative SPD Functionality need to advertise itself. To be composable, it must be described using a semantic formalism, compliant with the pSHIELD semantic model. Thus any Innovative SPD Functionality applies for the **registration** both to the Semantic Database and to the Service Registry.

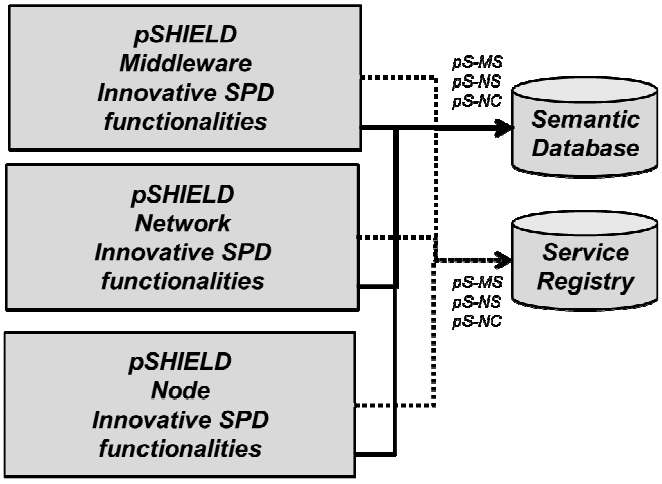

Figure 9 Innovative SPD Functionalities registration

4.3 Prototype objective

The pSHIELD WP5 prototype aims to demonstrate the main WP5 concepts by means of a software implementation based on OSGi (Open Service Gateway Initiative) technology. The WP5 SPD overlay intends to provide an homogeneous SPD level over heterogeneous embedded systems, applying a functional cross-layer approach. The SPD levels related to embedded systems' nodes, networks and middlewares are measured, abstracted and controlled by the WP5 overlay.
In order to manage the complexity of such a solution the WP5 prototype concentrated on the main features provided by the SPD overlay:

- Embedded System node, network and middleware resources discovery;
- (Re)-Composition of node, network and middleware resources to provide the needed SPD level;
- Actuation and provisioning of the composition decision.

The best solution to design and develop a proof of concept demonstrator based on these requirements, is to choose an open service platform based on SOA (Service Oriented Architecture). In a SOA, everything is treated as a service. Each service has its interfaces, its requirements, its dependencies, its proper dynamic and static parameters. In a SOA vision even an embedded system can be described and treated as a composed service. Each sensor, each node, each protocol, each resource can be described as a service, can be discovered, composed and delivered to a requestor.
In our vision the request is represented by a needed SPD level, described using the metrics introduced in WP2. Each available embedded system resource is described as a service and is characterized by a specific SPD level, measured using the metrics defined in WP2.

4.4 The OSGi framework

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

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

More in particular we decided to use the open source Knopflerfish OSGi service platform. Knopflerfish (hereafter referred as to KF) is a component-based framework for Java in which units of resources called bundles can be installed. Bundles can export services or run processes, and have their dependencies managed, such that a bundle can be expected to have its requirements managed by the container. Each bundle can also have its own internal classpath, so that it can serve as an independent unit, should that be desirable. All of this is standardized such that any valid Knopflerfish bundle can be installed in any valid OSGi container (Oscar, Equinox or any other).
Basically, running OSGi is very simple: one grabs one of the OSGi container implementations (Equinox, Felix, Knopflerfish, ProSyst, Oscar, etc.) and executes the container's boot process, much like one runs a Java EE server. Like Java EE, each container has a different startup environment and slightly different capabilities. The KF environment can be downloaded here: http://www.knopflerfish.org/
The KF start-up environment is shown below:

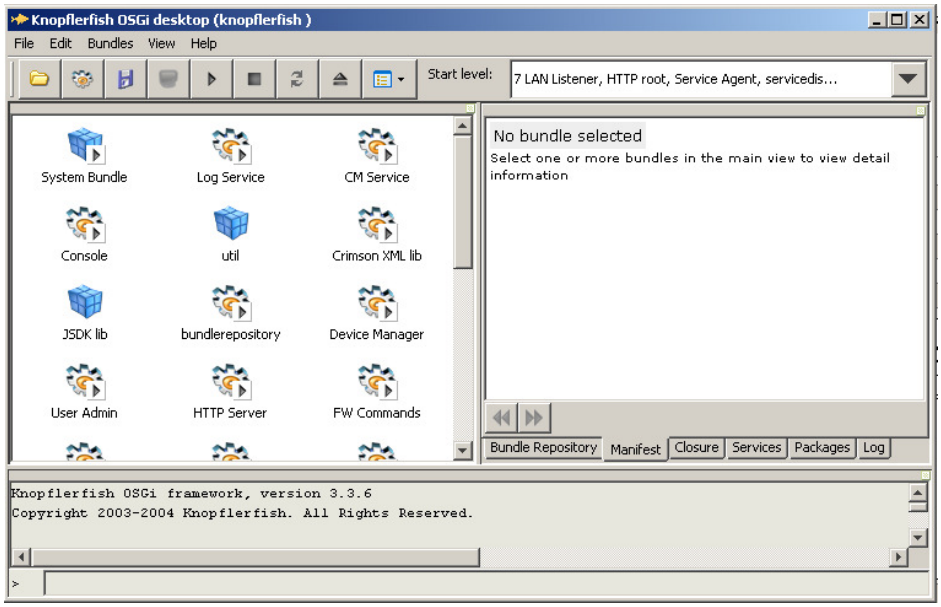

Figure 10 Knopflerfish start-up environment

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

4.5 Prototype Architecture

The prototype architecture derives directly from the architecture described in the previous section. Each pSHIELD component is mapped into an OSGi bundle and, when needed, decoupled into a composition of interoperating bundles each providing a specific functionality. This modular approach simplify the design, development and debugging of the whole system. Even the Innovative SPD Functionalities have been implemented as OSGi bundles. Each OSGi bundle has its own dependencies, provides a set of functionalities, requires a set of functionalities and is characterized by a specific SPD level. Each bundle can be registered in the Service Registry to advertise itself, to maintain updated its status in order to be discovered. Each bundle can also store its description in the Semantic Database, to be semantically composed. Each bundle interfaces the rest of the architecture providing a set of functionalities and requiring a set of functionalities, exactly as a software component does. More in particular each bundle is decoupled into two parts: the interfacing part (API) and its implementation part (IMPL). This separation between API and IMPL ease the substitution at runtime of a specific bundle, to change from one implementation to another. This substitution can be due, as an example, to the necessity to strengthen the SPD level of a specific functionality.

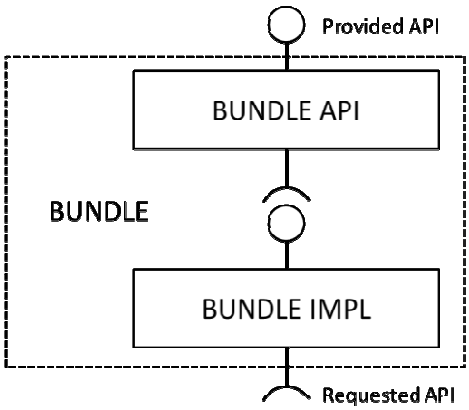

Figure 11 Bundle architecture

Applying for a top-down design approach, the Core SPD Services can be mapped in the following way:

- The Discovery and Composition are two separate bundles;
- The Orchestration is represented by the OSGi framework and orchestrate also the Discovery and Composition bundles.

To consider the interaction of the middleware layer with the rest of the architecture, the following additional bundles can be considered:

- The Service Registry bundle;
- The Semantic DB bundle;
- The SPD Security Agent bundle belonging to the pSHIELD Overlay layer;
- The pSHIELD Node, Network and Middleware Adapter that could be grouped into a single Adapter bundle.

The high level prototype architecture maps perfectly the Figure 4, as depicted in the following figure:

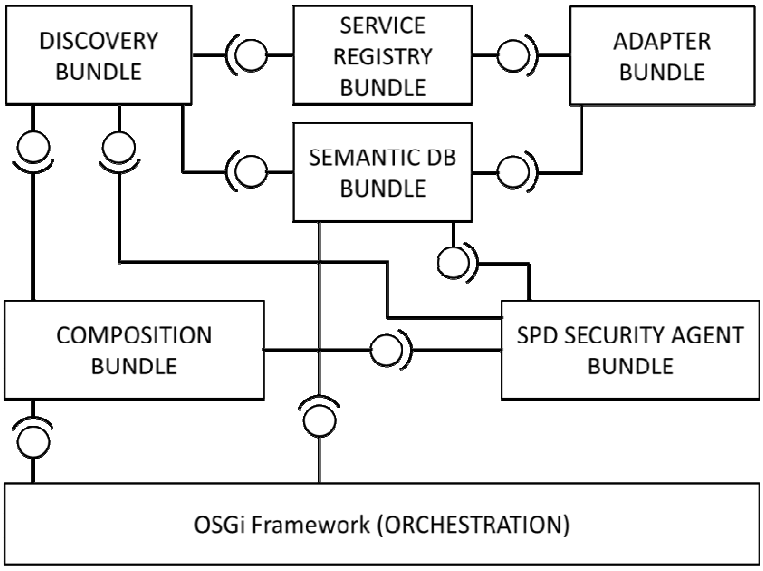

Figure 12 High level Core SPD Services prototype architecture

Let see in detail the structure of each bundle of the high level Core SPD Services prototype architecture.

4.5.1 Discovery Bundle

The discovery bundle structure is depicted in the following figure:

As explained in the previous sections, the Discovery Bundle is composed by the following bundles:

- **Discovery Engine Bundle**: it is in charge to handle the queries coming from the *IGenericDiscovery()* interface. The Discovery Engine Bundle manages the whole discovery process and activates the different functionalities of the Discovery service. It calls the *IQueryPreprocessor()* interface to enrich semantically and contextually the query. After that the query is sent to the different underlying discovery protocols, by means of the *IServiceDiscovery()* interface, to harvest over the interconnected systems all the available SPD components. Finally the list of discovered services is sent to the Filter Engine Bundle using the *IServicesFilter()* interface to discard those components not matching with the enriched query.
- **Query Preprocessor Bundle:** it is in charge to enrich the query sent by the Discovery Engine with semantic information related to the peculiar context. The query pre-processor can be configured by the SPD Security Agent to take care of the current environmental situation using the *IConfigureContext()* interface;
- **Discovery Protocol Bundle:** it is in charge to securely discover all the available SPD components description stored in the Service Registry Bundle, using a the *findServices()* interface;
- **Filter Engine Bundle:** it is in charge to semantically match the query with the descriptions of the discovered SPD components. In order to perform the semantic filtering, the Filter Engine can retrieve from the Semantic DB the information associated to the SPD components, by means of the *getOntology()* interface.

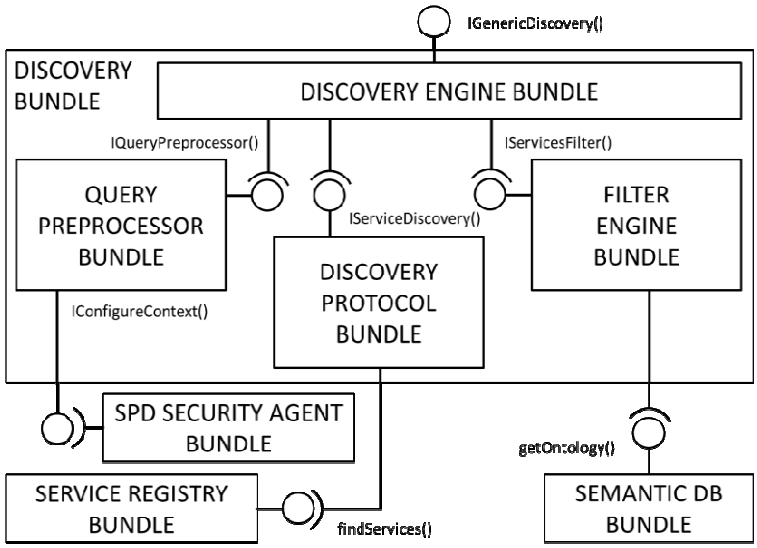

Figure 13 Discovery Bundle structure

4.5.2 Service Registry Bundle

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

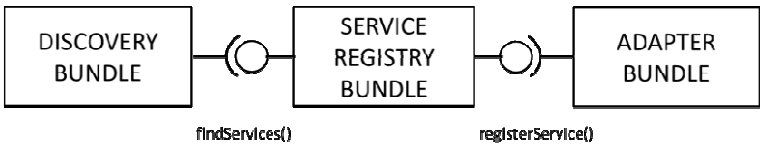

Figure 14 Service Registry Bundle

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

4.5.3 Adapter Bundle

The Adapter Bundle structure is depicted in the following figure:

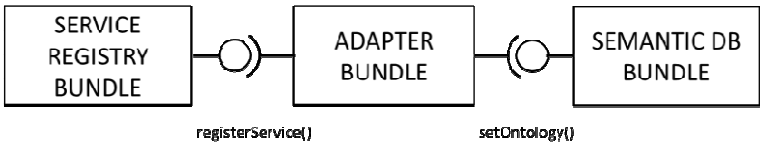

Figure 15 Adapter Bundle

- **Adapter Bundle**: it represents a generic (Node, Network or Middleware) pSHIELD Adapter for any type of legacy SPD functionality. The Adapter Bundle is in charge to:
  1. Provide an Innovative SPD functionality interacting with the underlying legacy services, capabilities and resources;

2. register the provided Innovative SPD Functionality in the Service Registry using the *registerService()* interface;
3. publish the semantic description of the Innovative SPD Functionality in the Semantic DB using the *setOntology()* interface;

4.5.4 Semantic DB Bundle

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

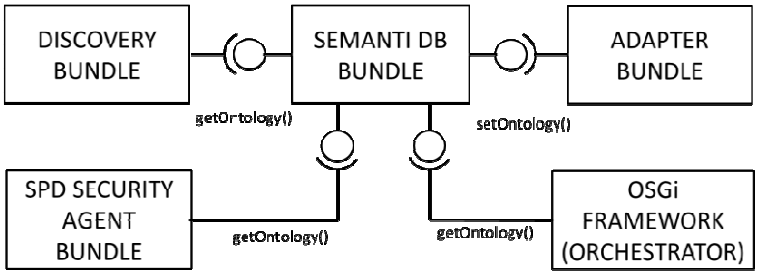

Figure 16 Semantic DB Bundle

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

4.5.5 Composition Bundle

The Composition Bundle structure is depicted in the following figure:

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

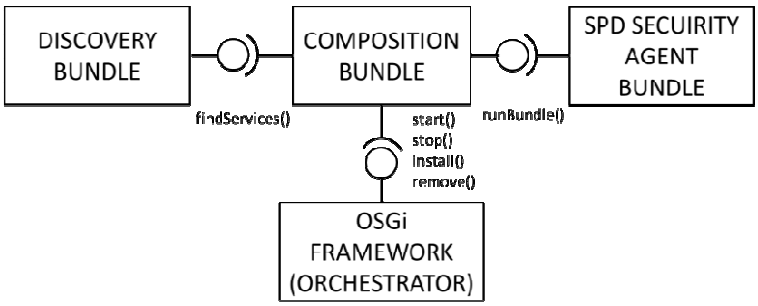

Figure 17 Composition Bundle

4.5.6 SPD Security Agent Bundle

The SPD Security Agent Bundle structure is depicted in the following figure:

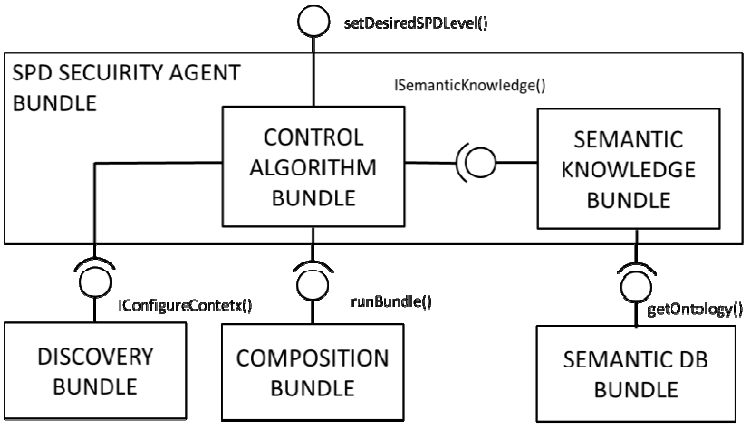

Figure 18 SPD Security Agent Bundle

As explained in the previous sections, the SPD Security Agent is composed by the following bundles:

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

4.6 Deployment details

The prototype infrastructure has been deployed into a real OSGi framework. A screenshot of the OSGi control panel is reported below:

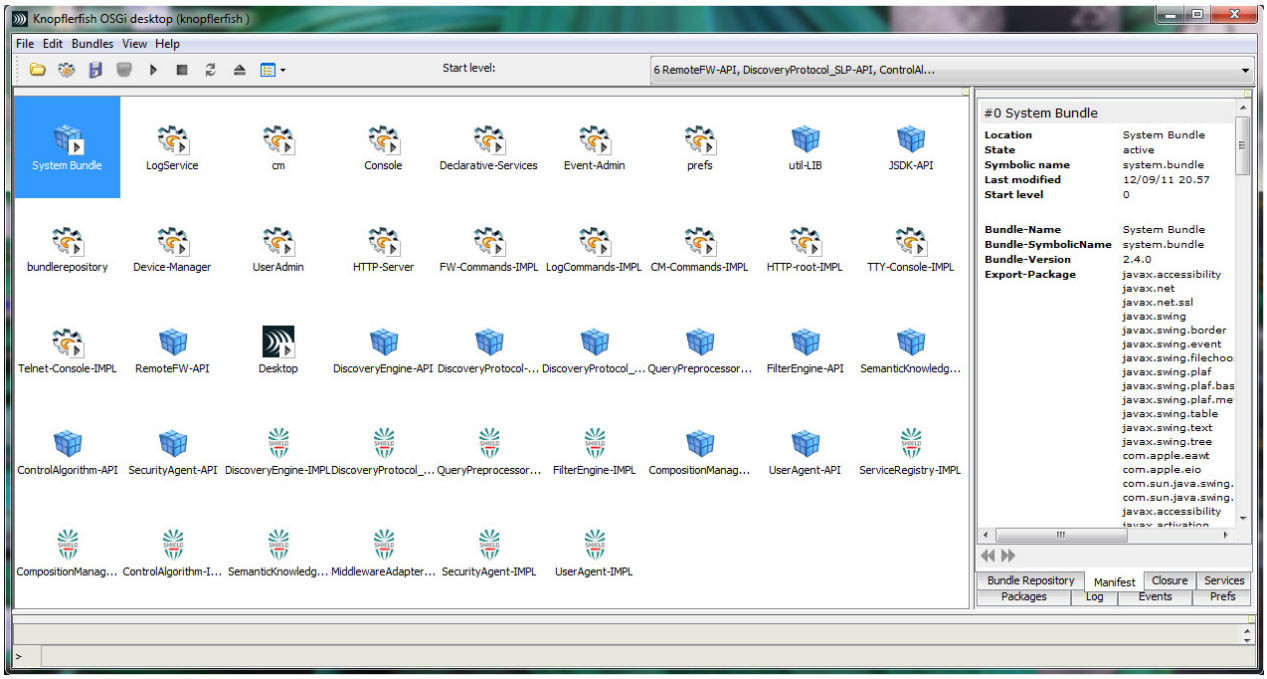

Figure 19 OSGI Environment

In this screenshot all the above introduced bundles are shown correctly running in a OSGi environment. The Core SPD Services prototype will be used to setup the pSHIELD pilot Demonstrator by adding proper pSHIELD Adapters representing meaningful components of the Railway Application Scenario.

In the following section some concepts on semantics (widely approached in D5.3) will be reported since they strictly interact with the Middleware.
Service Oriented Architectures and semantic web services are indeed one of the possible means of implementation, in a real environment, of the pSHIELD Middleware functionalities.
Particular attention will be devoted to the state of the art in semantic composition, as introduction to what will be presented in the rest of the deliverable as pSHIELD candidate solution to composition problem.

5 Trends in Semantic Web Services Technologies

Targeting new SPD middleware and overlay funtionalites prototypes related to Semantic Web Srvices (SWS) is requiring a deep understing of the trends in SWS technologies. Web services [8] have added a new level of functionality to the current Web by taking a first step towards seamless integration of distributed software components using web standards.

Problem statement: The Web service technologies around SOAP (XML Protocol Working Group, 2003), WSDL and UDDI operate at a syntactic level and, therefore, although they support interoperability (i.e., interoperability between the many diverse application development platforms that exist today) through common standards, they still require human interaction to a large extent: The human programmer has to manually search for appropriate Web services in order to combine them in a useful manner, which limits scalability and greatly curtails the added economic value of envisioned with the advent of Web services.

Recent research aimed at making web content more machine-processable, usually subsumed under the common term Semantic Web [9] are gaining momentum also, in particular in the context of Web services usage. **The semantic markup shall be exploited to automate the tasks of Web service discovery, composition and invocation, thus enabling seamless interoperation between them while keeping human intervention to a minimum** [10].

The convergence of semantic Web with service oriented computing is manifested by SWS technology. It addresses the major challenge of automated, interoperable and meaningful coordination of Web Services to be carried out by intelligent software agents.

SWS technologies were proposed in order to pursue the vision of the semantic Web presented in [9] whereby intelligent agents would be able to exploit semantic descriptions in order to carry out complex tasks on behalf of humans [11]. This early work on SWS was the meeting point between semantic Web, Agents, and Web services technologies. Gradually, however, research focussed more prominently on combining Web services with semantic Web technologies in order to better support the discovery, composition, and execution of Web services, leaving aspects such as systems' autonomy more typical of agent--based systems somewhat aside [12].

In one of the seminal papers on SWS, McIlraith et al. proposed the usage of Semantic Web technologies in order to markup Web services to make them machine-interpretable and accordingly facilitate automatic Web service discovery, execution, composition, and interoperability [11].

Here, we will focus on the trends around SWS and research and development, as well as standardization efforts on semantic Web services. Activities related to semantic Web services have involved developing conceptual models or ontologies, algorithms and engines that could support machines in semi-automatically or automatically discovering, selecting, composing, orchestrating, mediating and executing services. We will provide an overview of the area after nearly a decade of research.We will present the main principles and conceptual models proposed thus far including OWL-S, WSMO [12].

These four specifications related to Semantic Web Services were submitted to the World Wide Web Consortium (W3C) in 2004–2005.

1. OWL-S [23]
2. Semantic Web Services Framework (SWSF) [26]
3. Web Service Semantics (WSDL-S) [25]
4. Web Service Modeling Ontology (WSMO) [24]

For the most part, it is unclear now whether these specifications compete against, complement, or supersede one another. There are ongoing integration efforts. For instance, WSMO uses WSDL-S as its Service Grounding mechanism. Also, there is a W3C Semantic Web Services Interest Group that is chartered with discussing the integration of this work [111]. A clear understanding of the relationship among these four submissions and the types of problems in which they can be applied is necessary for the adoption of Semantic Web Services in general.

Investigating SWS technologies and making comaparison between them we do not believe it is a matter of deciding who the winner is, but rather how these different efforts can collaborate in producing a

standard that encompasses the benefits of each or provides clear instructions for use and integration into semantic solutions. Semantic Web services are an ongoing, dynamic research field. Nevertheless, they have the potential of providing a foundational pillar for the next generation of Web technology and they fit in our research effort to investigate how these technologies will enhance SPD middleware and overlay functionalities, as emphasized by the submission of WSMO and OWL-S to the W3C as starting points for upcoming standardization efforts.

The Semantic Web [54] approach refers to the idea of making the overwhelming amount of data on the Web machine-processable. This shall be achieved by annotating web content with consensual formalizations of the knowledge published, often referred to under the common term of *ontologies*. Language proposals for describing ontologies include the Resource Description Framework (RDF) [55], RDF Schema [56], the Web Ontology Language (OWL) [57] and the Web Service Modeling Language (WSML) family of languages [58].

The semantic Web working group of the W3C develops technologies and standards for the semantic description of the Web. The goal of these efforts is to make the Web understandable by machines and to increase the level of autonomous interoperation between computer systems [99]. Several standards and languages were already introduced to address this objective. The most relevant are the Resource Description Framework (RDF) [102], and the Web Ontology Language (OWL) [103], [101], [104], [105].

However, the goal of what is called SWS [59] is the fruitful combination of Semantic Web technology and Web services. By using ontologies as the semantic data model for Web Service technologies Web Services have machine-processable annotations just as static data on the Web. Semantically enhanced information processing empowered by logical inference eventually shall allow the development of high quality techniques for automated discovery, composition, and execution of Services on the Web, stepping towards seamless integration of applications and data on the Web. Two relevant initiatives have to be considered in the context of Semantic Web services. Chronologically, the first one is OWL-S [60], an upper level ontology for describing Web services, specified using OWL.

Questioning several deficiencies in this model [61], there is proposed a new framework for Semantic Web Services by the Web Service Modeling Ontology (WSMO) [62] which refines and extends the Web Service Modeling Framework (WSMF) [63] to a meta-ontology for Semantic Web services. **WSMF defines a rich conceptual model for the development and the description of Web services based on two main requirements: maximal decoupling and strong mediation. WSMO is accompanied by a formal language, the Web Service Modeling Language (WSML) that allows one to write annotations of Web services according to the conceptual model. Also an execution environment (WSMX) [64] for the dynamic discovery, selection, mediation, invocation, and inter-operation of Semantic Web services based on the WSMO specification is under development. [53]**.

OWL-S [110] is ontology for semantically describing services. The major difference to WSMO is that OWL-S does not consider the resolution of heterogeneity explicitly. While WSMO includes mediators as one of its key conceptual elements, OWL-S assumes that heterogeneity will be overcome by the underlying service infrastructure. For a complete comparison of both initiatives see [109]. METEOR-S[7] aims at integrating current (syntactical) Web Service initiatives for description, composition, etc. with Semantic Web technologies. The METEOR-S (METEOR for Semantic Web services) project was focused on the usage of semantics for the complete lifecycle of semantic Web processes, namely, annotation, discovery, composition, and execution. However, METEOR-S does not provide a conceptual model for the description of business services and does not specifically address the integration of heterogeneous businesses.

The Semantic Web Services Initiative (SWSI) [91] was an initiative of academic and industrial researchers, which has been composed to create infrastructure that combines Semantic Web and Web Services to enable the automation in all aspects of Web services. In addition to providing further evolution of OWL-S [92], SWSI will also be a forum for working towards convergence of OWL-S with the products of the WSMO [33] / WSML [94] / WSMX [95] research effort. WSMO is a complete ontology for the definition of Semantic Web Services. It follows the WSMF as a vision of Semantic Web Services. WSML is a family of languages that allow Semantic Web Service designers to define Semantic Web Services in a formal language. The WSMX provides a standard architecture for the execution of Semantic Web Services. Besides these major standards and initiatives, there are two ongoing projects being developed in the US, the LSDIS METEOR-S project [96], and in Europe, the DERI SWWS project [97], [98].

[7] http://lsdis.cs.uga.edu/Projects/METEOR-S/

5.1 OWL Web Ontology Language for Services (OWL-S)

This section is providing some details on the OWL-S, which is the result of a collaborative effort by researchers at several universities and organizations, including BBN Technologies, Carnegie Mellon University, Nokia Research Centre, Stanford University, SRI International, USC Information Sciences Institute, University of Maryland, Baltimore County, University of Toronto, Vrije Universiteit Amsterdam, University of Southampton, De Montfort University and Yale University.

The first approach for Semantic Web services has been provided by OWL-S [73, 74]. Using OWL as the description language, OWL-S defines an upper ontology for semantically describing Web services that is comprised of three top-level elements: A service in OWL-S is described by means of three elements, as shown in **Errore. L'origine riferimento non è stata trovata.**.

1. The ***Service Profile*** describes what the service does. It explains what the service accomplishes, details limitations on its applicability and quality of service, and specifies requirements the service requester must satisfy to use it successfully. This information is used by consumers during the discovery of the service.

2. The ***Service Process Model*** describes how to use the service. It details the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step-by-step processes leading to those outcomes.

3. The ***Service Grounding*** specifies the details of how to access/invoke a service. It includes communication protocol, message formats, serialization techniques and transformations for each input and output, and other service-specific details such as port numbers used in contacting the service [19].

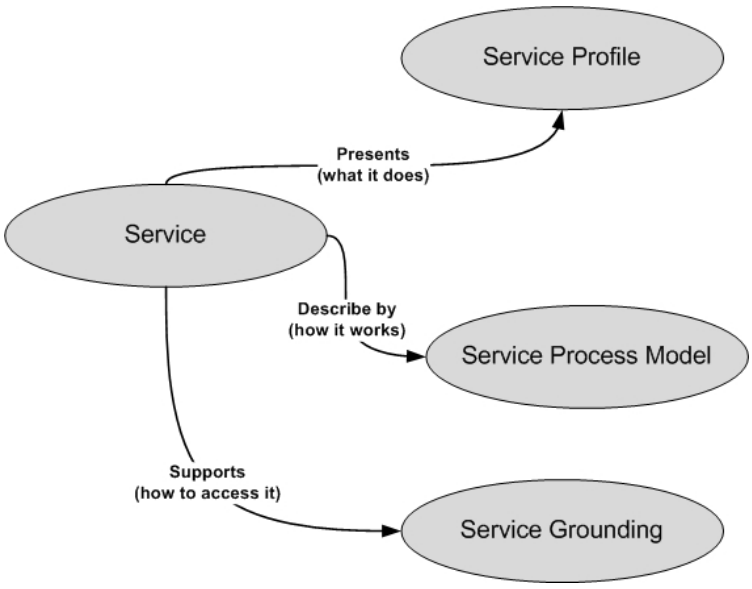

Figure 20 OWL-S Service Description Elements

Therewith, OWL-S provides a model for semantically describing Web services and serves as a basis for various research and development activities on Semantic Web service technologies [75]. However, OWL-S is criticized for conceptual weaknesses and incompleteness: the meaning of the description elements is not clearly defined and thus used ambiguously, leading to misinterpretations and incompatible service descriptions; furthermore, although OWL-S allows other languages like KIF and SWRL for process descriptions besides OWL, their formal intersection is not defined, hence a coherent formalism for semantically describing Web services is not provided [61]. However, the OWL-S provides developers with a strong language to describe the properties and capabilities of Web Services in such a way that the descriptions can be interpreted by a computer system in an automated manner [19]. In the following, we briefly summarize the underlying standard ontology language OWL and then present each of the main elements of OWL-S service descriptions.

The information provided by an OWL-S description includes

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

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

Consequently, the emerging concept of Semantic Web services aims at providing more sophisticated Web Service technologies along with support for the Semantic Web. Mentioned first in [11] and [32], **Semantic Web services shall utilize ontology's as the underlying data model in order to support semantic interoperability between Web services and its clients and apply semantically enabled mechanisms for automated discovery, composition, conversation, and execution of Web Services. Therefore, exhaustive description frameworks are required that define the semantic annotations of Web services needed for automatically determine their usability**.

OWL-S is an upper ontology used to describe the semantics of services based on the W3C standard ontology OWL and is grounded in WSDL. It has its roots in the DAML Service Ontology (DAML-S) released in 2001, and became a W3C candidate recommendation in 2005. OWL-S builds on top of OWL and consists of three main upper ontologies: the Profile, the Process Model, and the Grounding (Figure 3.17).

The OWL-S initiative [65] uses OWL to develop an ontology of services, covering different aspects of Web services, among them functionality. To describe their functionality, services are viewed as processes that (among other things) have pre-conditions and effects. However, the faithful representation of the dynamic behavior of such processes (what changes of the world they cause) is beyond the scope of a static ontology language like OWL. In AI, the notion of an action is used both in the planning and the reasoning about action communities to denote an entity whose execution (by some agent) causes changes of the world (see e.g. [67, 70]). Thus, it is not surprising that theories developed in these communities have been applied in the context of Semantic Web services. For example, [11, 39] use the situation calculus [67] and GOLOG [68] to formalize the dynamic aspects of Web services and to describe their composition. In [65], OWL-S process models are translated into the planning language of the HTN planning system SHOP2 [69], which are then used for automatic Web service composition.

5.1.1 OWL-S Discovery and Execution Elements

By itself, OWL-S is a language for the markup of Web Services. It becomes useful when there are tools to exploit Web Services described using OWL-S constructs.

An example of an OWL-S toolkit is CMU's OWL-S Development Environment (CODE), created by Carnegie Mellon University's Intelligent Software Agents Lab. CODE supports the complete OWL-S Web Services development process—from the generation of OWL-S descriptions (from Java code or WSDL documents) to the deployment and registration of the service. CODE is implemented as an Eclipse plug-in that supports activities for service providers and client application developers[8]. In addition to tools for the description of services, CODE includes the OWL-S Matchmaker and the OWL-S Virtual Machine (VM) elements. The OWL-S Matchmaker serves as a "catalog" of services defined using OWL-S. Service providers register OWL-S descriptions of services with the OWL-S Matchmaker. Client applications can query the OWL-S Matchmaker with an ontological description of the desired inputs and outputs. The

[8] Eclipse is an open source community engaged in providing a vendor-neutral software development platform. Read more about Eclipse at http://www.eclipse.org/

OWL-S Matchmaker matches the request with its catalog of services and returns a ranked list of services that most closely match the request.

The OWL-S VM is used to invoke services using OWL-S. After the client application selects a service from the ranked list of services, it formulates its request using the format specified by the OWL ontology and sends the request to the OWL-S VM. Using Extensible Style Language Transformations (XSLT)[9] present in the Service Grounding element, the OWL-S VM reformats the request to match the format required by the service. Then, it invokes the service on behalf of the client. When it receives a response to that step, the OWL-S VM uses another XSLT transformation in the Service Grounding element to reformat the response into a format matching that of the ontology. Finally, the OWL-S VM sends the response back to the client application. In this manner, the client application does not need to know anything about how to interact with the actual service; the OWL-S VM acts as a mediator for the request and response.

Both the OWL-S Matchmaker and the OWL-S VM elements have an Application Programming Interface (API) for Java applications to discover and invoke services [17].

Fortunately, the Intelligent Software Agents Lab at Carnegie Mellon University has played a critical role in the development of OWL-S from its very inception [112]. This group developed CODE. Thus, CODE was selected as the development environment, and support was graciously provided by Naveen Srinivasan of the Intelligent Software Agents Lab.

5.1.1.1 Service Profile

The OWL-S profile ontology is used to describe what the service does, and is meant to be mainly used for the purpose of service discovery. An OWL-S service profile or signature encompasses its functional parameters, i.e. hasInput, hasOutput, precondition and e.ect (IOPEs), as well as non-functional parameters such as serviceName, serviceCategory, qualityRating, textDescription, and meta-data (actor) about the service provider and other known requesters. Please note that, in contrast to OWL-S 1.0, in OWL-S 1.1 the service IOPE parameters are de.ned in the process model with unique references to these definitions from the profile.

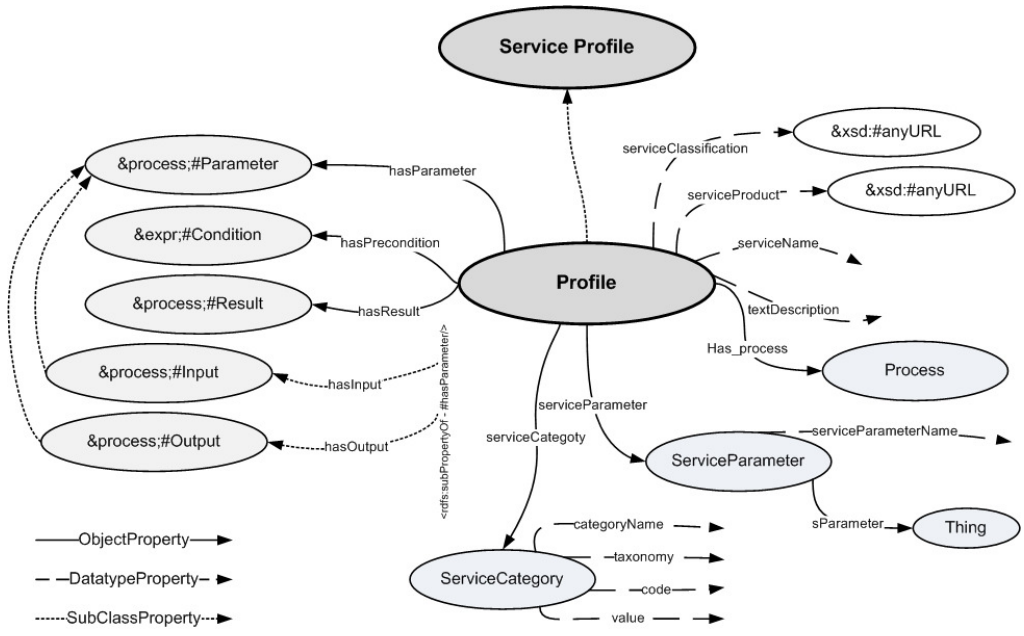

Figure 21 OWL-S service profile structure

Inputs and outputs relate to data channels, where data flows between processes. Preconditions specify facts of the world (state) that must be asserted in order for an agent to execute a service. Effects characterize facts that become asserted given a successful execution of the service in the physical world

[9] In its simplest definition, *XSLT* is a language used to transform XML documents into other XML documents. Currently, XSLT is the only type of transformation supported by the OWL-S VM

(state). Whereas the semantics of each input and output parameter is defined as an OWL concept formally specified in a given ontology, typically in decidable OWL-DL or OWL-Lite, the preconditions and effects can be expressed in any appropriate logic (rule) language such as KIF, PDDL, and SWRL. Besides, the profile class can be sub classed and specialized, thus supporting the creation of profile taxonomies which subsequently describe different classes of services.

5.1.1.2 Service Process Model

An OWL-S process model describes the composition (choreography and orchestration) of one or more services that is the controlled enactment of constituent processes with respective communication pattern. In OWL-S this is captured by a common subset of workflow features like split + join, sequence, and choice Figure 22.

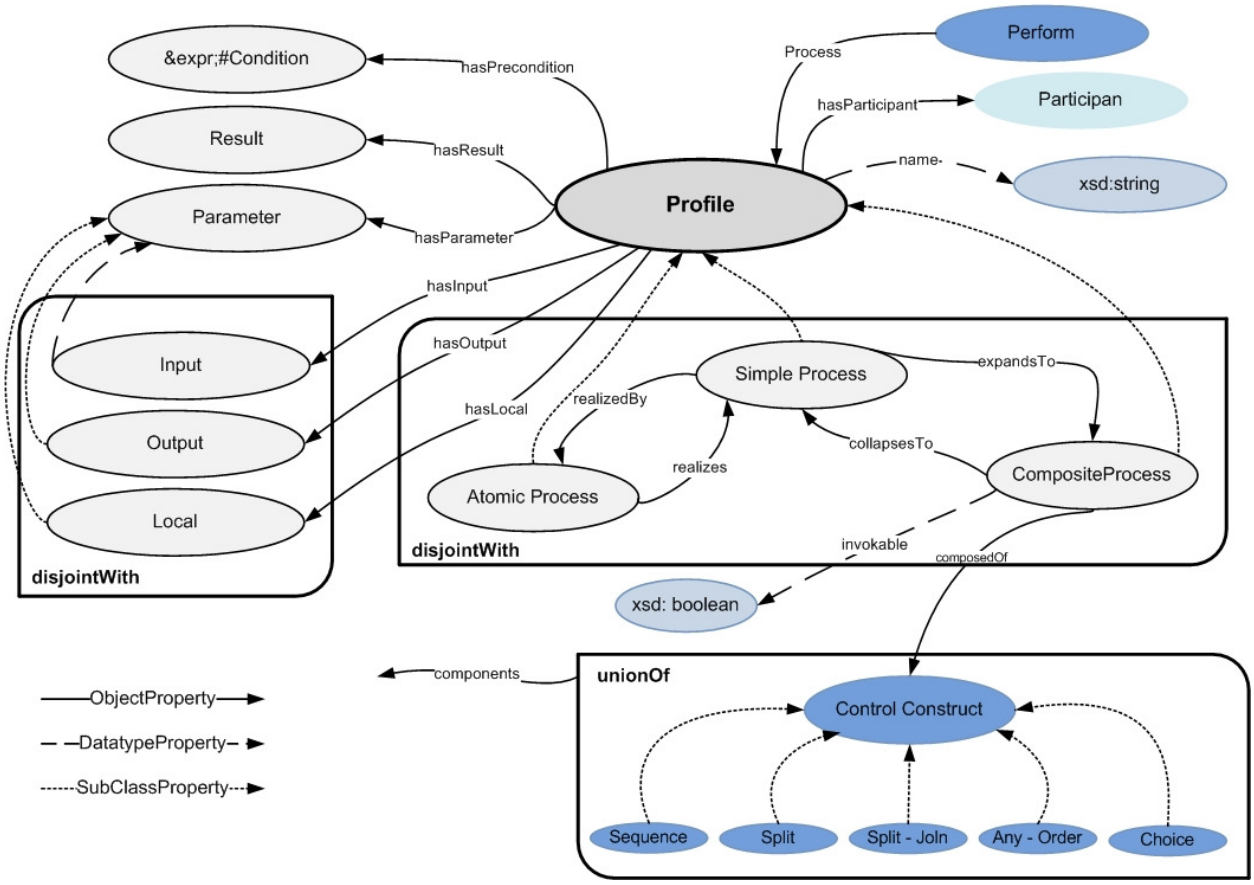

Figure 22 OWL-S service process model

Originally, the process model was not intended for service discovery but the profile by the OWL-S coalition. More concrete, a process in OWL-S can be atomic, simple, or composite. An atomic process is a single, black-box process description with exposed IOPEs. Simple processes provide a means of describing service or process abstractions which have no specific binding to a physical service, thus have to be realized by an atomic process, e.g. through service discovery and dynamic binding at runtime, or expanded into a composite process.

Composite processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. This process workflows are constructed using a number of different control flow operators including Sequence, Unordered (lists), Choice, If-then-else, Iterate, Repeat-until, Repeat-while, Split, and Split + Join. In OWL-S 1.1, the process model also speci.es the inputs, outputs, preconditions, and effects of all processes that are part of a composed service, which are referenced in the profiles of the respective services. An OWL-S process model of a composite service can also specify that its output is equal to some output of one of its sub processes whenever the composite process gets instantiated.

Moreover, for a composite process with a Sequence control construct, the output of one sub process can be de.ned to be an input to another sub process (binding).

Unfortunately, the semantics of the OWL-S process model are left undefined in the official OWL-S documents. Though there are proposals to specify these semantics in terms of, for example, the situation calculus, and the logic programming language GOLOG based on this calculus [27].

5.1.1.3 Service Grounding

The grounding of a given OWL-S service description provides a pragmatic binding between the logic-based and XMLS-based service definitions for the purpose of facilitating service execution. Such a grounding of OWL-S services can be, in principle, arbitrary but has been exemplified for grounding in WSDL to pragmatically connect OWL-S to an existing Web Service standard Figure 23.

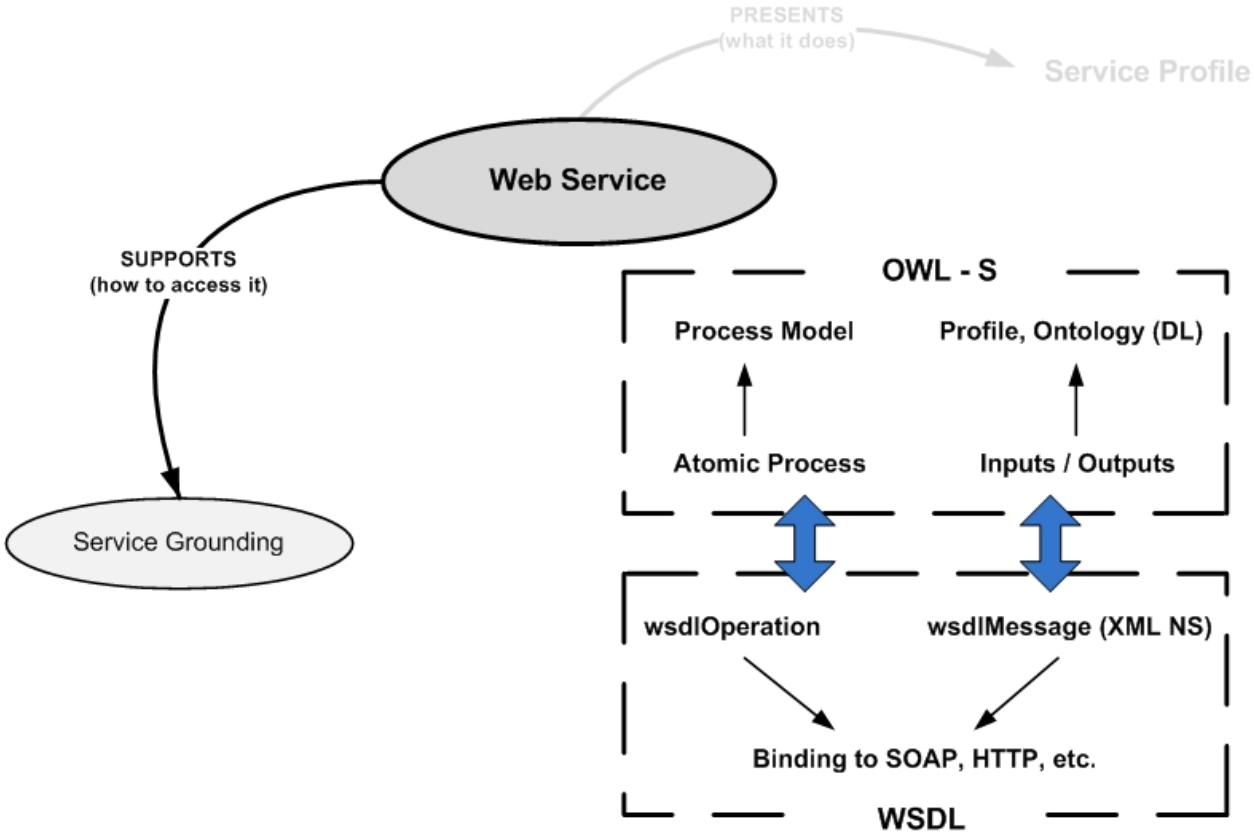

Figure 23 Grounding of OWL-S in WSDL

In particular, the OWL-S process model of a service is mapped to a WSDL description through a thin (incomplete) grounding: Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of respectively named XML data types of corresponding input and output messages. Unlike OWL-S, WSDL cannot be used to express pre-conditions or effects of executing services. Any atomic or composite OWL-S service with grounding in WSDL is executable either by direct invocation of the (service) program that is referenced in the WSDL file, or by a BPEL engine that processes the WSDL groundings of simple or orchestrated Semantic Web Services.

5.1.2 Software Support

One prominent software portal of the semantic Web community is SemWebCentral9 developed by InfoEther and BBN Technologies within the DAML program in 2004 with BBN continuing to maintain it today. As a consequence, it comes at no surprise that this portal o.ers a large variety of tools for OWL

and OWL-S service coordination as well as OWL and rule processing. Examples of publicly available software support of developing, searching, and composing OWL-S services are as follows. [18]
http://projects.semwebcentral.org/

- *Development* OWL-S IDE integrated development environment[10], the OWL-S 1.1 API[11] with the OWL-DL reasoner Pellet[12] and OWL-S editors.
- *Discovery* OWL-S service matchmakers OWLS-UDDI[13], OWLSM[14] and OWLS-MX[15] with test collection OWLS TC2.
- *Composition* OWL-S service composition planners OWLS-XPlan[16], GOAL[17].

The model problem experience has proven that regardless of its contributions to the semantics community, OWL-S is not ready to support the dynamic discovery, composition, and invocation of services, mostly due to the scarcity of tool support. Nonetheless, despite its problems, OWL-S has tremendous potential if given the proper resources and opportunities. Being able to define the inputs and outputs of a service in terms of ontology is a huge step towards dynamic discovery, composition, and invocation without user intervention. Unfortunately, funding—which is what makes technologies real—has stopped for OWL-S tool development. Additional investment in completing and debugging CODE (or other tools that are as further along as CODE) would demonstrate the feasibility of the technology and encourage potential adoption from industry that in turn would provide feedback for further improvements. With development, OWL-S could enable applications to dynamically discover, compose, and invoke new services to solve problems and gain information in ways that were not available when those applications were created. OWL-S has the capability to embed semantic meaning into the collections of services available in services-oriented computing environments, which will allow applications to be developed without knowledge of specific services that may or may not be available. That capability could also make SOAs more robust and flexible. Collections of services that are defined using OWL-S would allow applications to be tolerant of faults in a dynamic and transparent way by simply accessing other services with similar semantic meanings. OWL-S could also enable services to be registered and removed at runtime dynamically without causing downtime in client applications.

[10] http://projects.semwebcentral.org/projects/owl-s-ide/
[11] http://projects.semwebcentral.org/projects/owl-s-api
[12] http://projects.semwebcentral.org/projects/pellet/
[13] http://projects.semwebcentral.org/projects/mm-client/
[14] http://projects.semwebcentral.org/projects/owlsm/
[15] http://projects.semwebcentral.org/projects/owls-mx/
[16] http://projects.semwebcentral.org/projects/owls-xplan/
[17] http://www.smartweb-project.de

5.2 Web Service Modeling Ontology (WSMO)

WSMO [28, 29, 76] is a member submission to W3C of an ontology that aims at describing all relevant aspects for the partial or complete automation of discovery, selection, composition, mediation, execution and monitoring of Web services. WSMO has its roots in the Web Service Modeling Framework (WSMF) and in Problem- Solving Methods [30], notably the Unified Problem Solving Method Development Language UPML [31], which have been extended and adapted in order to support the automation of the aforementioned tasks for manipulating Web services.

The Web Service Modeling Ontology (WSMO) aims at describing all relevant aspects related to general services which are accessible through a Web service interface with the ultimate goal of enabling the (total or partial) automation of the tasks (e.g., discovery, selection, composition, mediation, execution, monitoring, etc.) involved in both intra- and inter- enterprise integration of Web services. WSMO has its conceptual basis in the Web Service Modeling Framework (WSMF) [32], refining and extending this framework and developing a formal ontology and set of languages [10].

Taking the Web Service Modeling Framework WSMF as its conceptual basis [32], the WSMO project is an ongoing research and development initiative for defining a capacious framework for Semantic Web services along with a description language (the Web Service Modeling Language WSML [78]) and a reference implementation (the Web Service Execution Environment WSMX [77]).

In order to provide a detailed synopsis of the aims, challenges, and respective solutions for Semantic Web services, the following explains the design principles and element definitions of WSMO in detail.

5.2.1 WSMO Design Principles

Since Semantic Web services aim at turning the Internet from an information repository for human consumption into a world-wide system for distributed Web computing by combining Semantic Web technologies and Web services, WSMO, as any other framework for Semantic Web services description needs to integrate the *basic Web design principles*, the *Semantic Web design principles*, as well as *design principles for distributed, service oriented computing for the Web*. This section enumerates and discusses the design principles of WSMO.

5.2.2 WSMO Basic Concepts

WSMO defines the modeling elements for describing Semantic Web services based on the conceptual grounding set up in the Web Service Modeling Framework (WSMF) [63], wherein four main components are defined: ontologies, Web services, goals, and mediators.

WSMO inherits these four top elements, further refining and extending them. *Ontologies* represent a key element in WSMO since they provide (domain specific) terminologies for describing the other elements. They serve a twofold purpose: defining the formal semantics of the information, and linking machine and human terminologies.

Web services connect computers and devices using the standard Web-based protocols to exchange data and combine data in new ways. Their distribution over the Web confers them the advantage of platform independence. Each Web service represents an atomic piece of functionality that can be reused to build more complex ones. Web services are described in WSMO from three different perspectives: non-functional properties, functionality and behavior. *Goals* specify objectives that a client might have when consulting a Web service, i.e. functionalities that a Web service should provide from the user perspective. The coexistence of goals and Web services as non-overlapping entities ensures the decoupling between request and Web service. This kind of stating problems, in which the requester formulates objectives without regard to Web services for resolution, is known as *the goal-driven approach*, derived from *the AI rational agent approach*. *Mediators* describe elements that aim to overcome the mismatches that appear between the different components that build up a WSMO description. The existence of mediators allows one to link possibly heterogeneous resources. They resolve incompatibilities that arise at different levels:
– data level - mediating between different used terminologies, more specifically solving the problem of

ontology integration process level - mediating between heterogeneous communication patterns. This kind of heterogeneity appears during the communication between Web services.

This deliverable briefly introduced WSMO, one of the most salient efforts in the domain of Semantic Web services. Semantic Web Services are a key application area for Intelligent Agent Systems because of the necessity for semantic descriptions frameworks as a basis for such Intelligent Systems. Thus, WSMO and its formalization WSML provide the infrastructure for such systems.

Several implementations are already available or under development. The first version of the Web Service Execution Environment (WSMX) has been available since June 2004 at the Source Forge portal. Apart from WSMX which marks a reference architecture and implementation for a WSMO compliant execution environment there already exists several other ready-to-use implementations and tools for WSMO.

5.2.3 Top-level elements of WSMO

Following the key aspects identified in the Web Service Modeling Framework, WSMO identifies four top-level elements as the main concepts which have to be described in order to define Semantic Web Services:

Ontologies provide the terminology used by other WSMO elements to describe the relevant aspects of the domains of discourse. In contrast to mere terminologies that focus exclusively on syntactic aspects, ontologies can additionally provide formal definitions that are machine-processable and thus allow other components and applications to take actual meaning into account.

Web services represent computational entities able to provide access to services that, in turn, provide some value in a domain; Web service descriptions comprise the capabilities, interfaces and internal working of the service.

All these aspects of a Web service are described using the terminology defined by the ontologies.
Goals describe aspects related to user desires with respect to the requested functionality; again, Ontologies can be used to define the used domain terminology, useful in describing the relevant aspects of goals. Goals model the user view in the Web service usage process, and therefore are a separate top-level entity.

Finally, Mediators describe elements that handle interoperability problems between different WSMO elements. We envision mediators as the core concept to resolve incompatibilities on the data, process and protocol level, i.e. in order to resolve mismatches between different used terminologies (data level), in how to communicate between Web services (protocol level) and on the level of combining
Web services (process level).

5.2.3.1 Ontologies

In compliance to the vision of the Semantic Web, WSMO uses ontologies as the underlying data model for Semantic Web services. This means that all resource descriptions and all information interchanged during collaboration execution is based on ontologies, thereby providing the basis for semantically enhanced information processing and ensuring semantic interoperability between Semantic Web services. In accordance to the AI-theory of ontologies [80], WSMO ontologies consist of the following elements: *Concepts* describe the entities of a domain that are characterized by *Attributes*; *Relations* describe associations between concepts, whereby subsumption and membership relationships define the taxonomic structure of ontology. An *Instance* is a concrete individual of a concept, and *Axioms* define constraints and complex aspects of the domain in terms of logical expressions. Regarding engineering methodologies developed for the Semantic Web [81], ontology design in WSMO demands and supports *modularization*, i.e. small-sized and concise ontologies, *decoupling*, i.e. distributed and multi-party ontology development and *ontology mediation* for resolving possibly occurring mismatches between loosely coupled ontologies for a specific usage scenario.

5.2.3.2 Web Services

WSMO defines a description model that encompasses that information needed for automatically determining the usability of a Web service. WSMO Web service description is comprised of four elements: (1) *non-functional properties*, (2) a *capability* as the functional description of the service; summarized as service interfaces, (3) a *choreography* that describes the interface for service consumption by a client, and (4) an *orchestration* that describes how the functionality of the service is achieved by aggregating other Web services. These notions describe the functionality and behavior of a Web service, while its internal implementation is not of interest.

5.2.3.3 Goals

In order to facilitate automated Web service usage and support ontological separation of user desires, service usage requests, and Web service descriptions, Goals in WSMO allow specifying objectives that clients - which can be humans or machines - wish to achieve. The general structure of WSMO Goal descriptions is similar to Web service descriptions. The client can specify the functionality expected in a *requested capability* that can consist of the same elements as Web service capabilities as discussed above. Also, a Goal can carry information on the expected behavior of an acceptable Web service in so-called *requested interfaces* that can define the excepted communication behavior for consuming a Web service with respect to its Choreography Interface as well as restrictions on other Web services aggregated in the orchestration of an acceptable Web service (e.g. only Web services are accepted that utilize a trusted payment facility). It is important to remark that Goal descriptions are defined from the client perspective, thereby decoupled from Web service descriptions.

5.2.3.4 Mediators

Mediation is concerned with handling heterogeneity, i.e. resolving possibly occurring mismatches between resources that ought to be interoperable. Heterogeneity naturally arises in open and distributed environments, and thus in the application areas of Semantic Web services. Hence, WSMO defines the concept of Mediators as a top level notion

Mediator-orientated architectures as introduced in [82] specify a mediator as an entity for establishing interoperability of resources that are not compatible a priori by resolving mismatches between them at runtime. The aspired approach for mediation relies on declarative description of resources whereupon mechanisms for resolving mismatches work on a structural, semantic level, in order to allow generic, domain independent mediation facilities as well as reuse of mediators. Concerning the needs for mediation within Semantic Web services, WSMO distinguishes three levels of mediation:

1. Data Level Mediation - mediation between heterogeneous data sources; within ontology-based frameworks like WSMO, this is mainly concerned with ontology integration.
2. Protocol Level Mediation - mediation between heterogeneous communication protocols; in WSMO, this mainly relates to choreographies of Web services that ought to interact.
3. Process Level Mediation - mediation between heterogeneous business processes; this is concerned with mismatch handling on the business logic level of Web services (related to the orchestration of Web services).

5.2.4 Semantic Web Service Technologies

Hence, the main working areas of Semantic Web service technologies for enabling automated Web service usage address the following aspects wherefore we outline most recent approaches below.

- Discovery and Selection: how to determine appropriate Web services for solving the goal of a client, and how to select the concrete Web service to be used in case several service can be used?
- Composition: if there does not exists a Web service that can completely satisfy a more complex goal, how to determine Web services that can be combined for solving the goal, and how to determine a suitable execution order for these services?
- Conversation Validation: given the behavior descriptions of Web services and clients that are ought to interact, how to determine whether the information interchange expected by each party can be achieved successfully?

- Mediation: how to resolve mismatches and heterogeneities that hamper Web services and clients to interoperate?
- Execution Support: how to manage and control execution of Web services, and how to ensure information interchange with respect to Web scale?

5.2.5 Discovery

Discovery within Web services is concerned with detecting suitable Web services for achieving a requester's objective, i.e. a goal. As outlined introductory, UDDI supports discovery in conventional Web service technology as follows. The service requester, which in this setting is expected to be a system developer, browses a UDDI repository, retrieving information on available Web services. Then, he manually inspects the usability of a Web service and integrates the respective technical service invocation into the target application. As this technology support appears to be unsatisfactory for automated Web service usage, Semantic Web service discovery aims at automatically determining the usability of a Web service by semantic matchmaking.

Referred to as functional discovery, the general approach is to inspect certain logical relationships between the semantic capability descriptions of requests (i.e. Goals) and Web services. If such a relationship holds, the Web service is considered to be usable for resolving the client's goal. On basis of several preceding works on semantic matchmaking [83], [84], the Web service discovery framework defined for WSMO [85] identifies five matchmaking notions as the core for functional discovery as shown in Figure 24. The important characteristic of these notions is that each one denotes a different logical relationship that has to hold for considering a Web service to be suitable for achieving a given Goal.

For instance, the Exact Match holds if and only if for each possible ontology instance that can satisfy the Web service holds that it also satisfies the Goal, and that there exists no possible ontology instance that satisfies only the Goal or the Web service. In contrast, the Intersection Match holds if there exists one possible instance that can satisfy both the Goal and the Web service. Hence, in order to precisely express client objectives with respect to discovery, WSMO Goals carry an additional non-functional property type Of Match denoting the matchmaking notion to be applied.

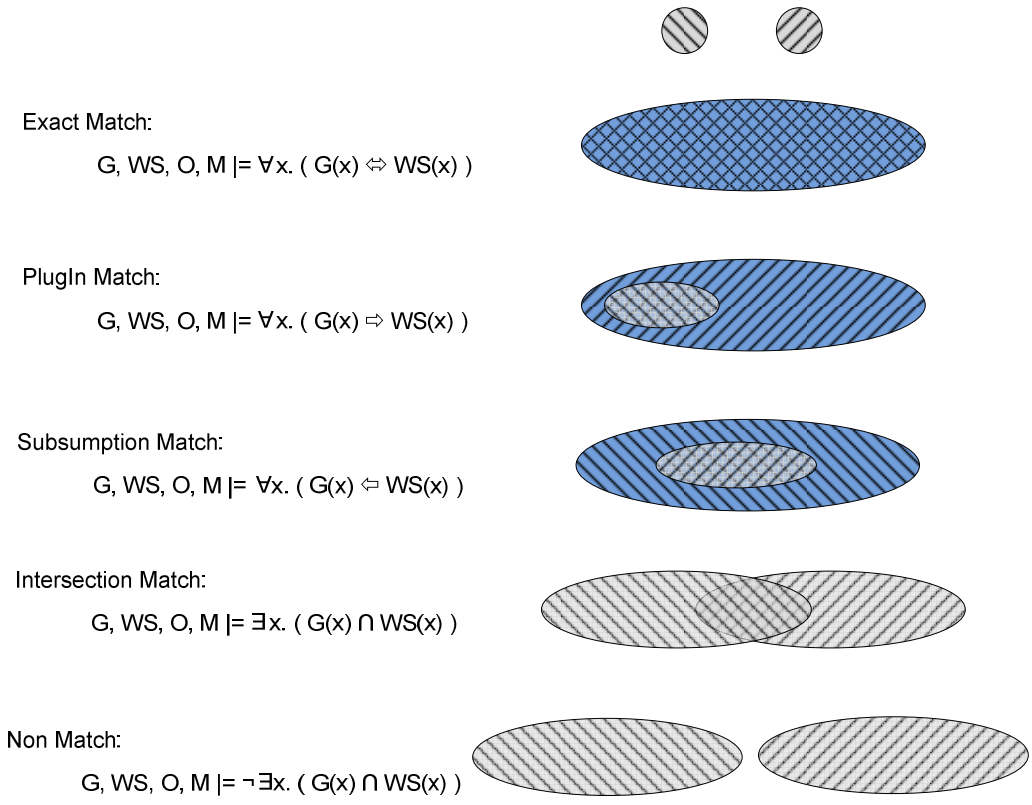

Figure 24 Semantic Discovery Matchmaking Notions

Several prototypical realizations for semantically enabled discovery are existing: [84] and [86] apply Description Logic reasoners, the approach in [87] is based on Frame- and Transaction Logic, and [88] use a FOL theorem proves as the technical platform for semantically enabled discovery. However, all these approaches are restricted to the specific reasoning support provided by the used tools. Hence, it is expected that these approaches will converge towards an integrated discovery framework for Semantic Web services that provides appropriate reasoning facilities[18].

[18] Description Logic and Logic Programming are the most prominent decidable subsets of First Order Logic; existing reasoners for each subset provide specific reasoning support. For discovery, as well as for several other reasoning tasks on the Semantic, facilities from both fractions are required. Hence, ongoing efforts aim at defining the maximal intersection of Description Logic and Logic Programming languages in order to facilitate integrated reasoning support. See [89] for further discussion.

5.3 IRS-III

The Internet Reasoning Service (*IRS-III*) is an infrastructure for publishing, locating, executing and composing Semantic Web services, organized according to the WSMO framework. *WSMO Studio* is a Semantic Web Service editor compliant with WSMO. The WSMO Studio will be available as a set of Eclipse plug-ins that will allow easy reusability and extension from 3rd parties. wsmo4j is a Java API and a reference implementation for building Semantic Web services applications compliant with WSMO. Like WSMX, it is also being developed as an Open Source project. The Semantic Web Fred [55] combines combine agent technology with WSMO, in order to provide advanced support for Semantic Web applications.

The IRS project (http://kmi.open.ac.uk/projects/irs) has the overall aim of supporting the automated or semi-automated construction of semantically enhanced systems over the internet. IRS-I supported the creation of knowledge intensive systems structured according to the UPML framework and IRS-II integrated the UPML framework with Web Service technology. IRS-III [33] has incorporated and extended the WSMO ontology [34] so that the implemented infrastructure allows the description, publication and execution of Semantic Web Services (SWS). The meta-model of WSMO describes four top level elements (in italics hence forth):

- *Ontologies*,
- *Goals*,
- *Web Services*, and
- *Mediator*s.

Ontologies provide the foundation for semantically describing data in order to achieve semantic interoperability and are used by the three other WSMO elements.

Goals define the tasks that a service requester expects a *web service* to fulfill. In this sense they express the service requester's intent. *Web services* represent the functional behavior of an existing deployed *Web Service*. The description also outlines how Web Services communicate (*choreography*) and how they are composed (*orchestration*).

Mediators describe the connections between the components above and represent the type of conceptual mismatches that can occur. In particular, WSMO provides four kinds of *mediator*s: *oo-mediators* link and map between heterogeneous ontologies; *ww-mediators* link *web services* to *web services*; *wg-mediators* connect *web services* to *goals*; *gg-mediators* link different *goals*.

IRS-III provides the representational and reasoning mechanisms for implementing the WSMO meta-model mentioned above in order to describe Web Services. Additionally, IRS-III provides a powerful execution environment which enables these descriptions to be associated to a deployed Web Service and instantiated during selection, composition, mediation and invocation activities. [35]

IRS (Internet Reasoning Service) is a framework to support Semantic Web Services, in which services can be described by their semantics, discovered, invoked and monitored. IRS-III consists of three main components, IRS Server, IRS Publisher and IRS Client. IRS Server stores and reasons with Web Service and Goal descriptions. IRS Publisher generates wrappers for programs as Web Services. Invocation of Web Services via Goal descriptions is supported by the IRS Client.

The notions of Goal and Mediator are particular characteristic of IRS-III and WSMO ontology. While a Web Service is a description of a method and concerned with the specification of mechanisms and execution, a Goal is a general description of a problem and concerned with describing a problem rather than mechanisms. As a result, Goals are suitable for describing a service for a user whose concern is not the technical details of the solution.

5.3.1 The IRS-III Framework

IRS-III is based on a distributed architecture composed of the IRS-III server, the publishing platforms and clients which communicate through the SOAP protocol, as shown in Figure 25. The server handles ontology management and the execution of knowledge models defined for WSMO. The server also

receives SOAP requests (through the API) from client applications for creating and editing WSMO descriptions of *goals*, *web services* and *mediators* as well as goal-based invocation. At the lowest level the IRS-III Server uses an HTTP server written in Lisp, which has been extended with a SOAP handler.
The publishing platforms allow providers of services to attach semantic descriptions to their deployed services and provide handlers to invoke services in a specific language or platform (Web Services WSDL, Lisp code, Java code, and Web applications). When a Web Service is published in IRS-III the information about the publishing platform (URL) is also associated with the *web service* description in order to be invoked. The IRS-III server is written in Lisp and is available as an executable file.

The publishing platforms are delivered as Java Web applications; and client applications use the Java API. [35]

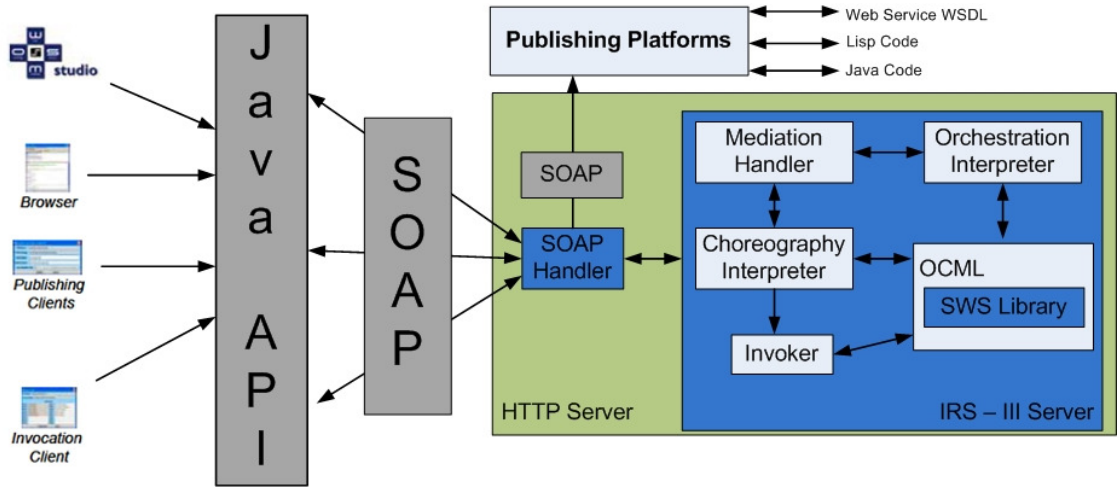

Figure 25 The IRS-III framework

The main components of IRS-III are explained in the following:

SWS Library – At the core of the IRS-III server is the SWS library where the semantic descriptions are stored using our representation language OCML. The library is structured into knowledge models for *goals*, *web services* and *mediators*. Domain *ontologies* and knowledge bases (instances) are also available from the library.

Choreography Interpreter – This component interprets the *grounding* and *guarded transitions* of the *choreography* description when requested by the mediation handler.

Orchestration Interpreter – This component interprets the workflow of the orchestration description when requested by the mediation handler.

Mediation Handler – The brokering activities of IRS-III including selection, composition and invocation are each supported by a specific mediation component within the mediation handler. These activities may involve executing a mediation service or mapping rules declared in a *mediator* description.

Invoker – The invoker component of the server communicates with the publishing platform, sending the inputs from the client and bringing the result back to the client.

5.4 Service Composition

SEMANTIC web service (SWS) composition process is generally performed when no available single or composite service can satisfy the required request and a combination which can satisfy request can be generated from available single or composite services.

Regarding service composition methods different techniques can be found in the bibliography. These techniques are usually divided into the categories of static (services to be composed decided at design time) and dynamic composition (services to be composed decided at runtime), or automatic (no user intervention) and manual composition (user-driven composition). The service composition approaches are categorized according to the research area they base their methods on. AI planning introduces the principles of Artificial Intelligence into service composition, Semantic Web approaches exploits the semantic characteristics of a service and Middleware approaches consider different middleware solutions.

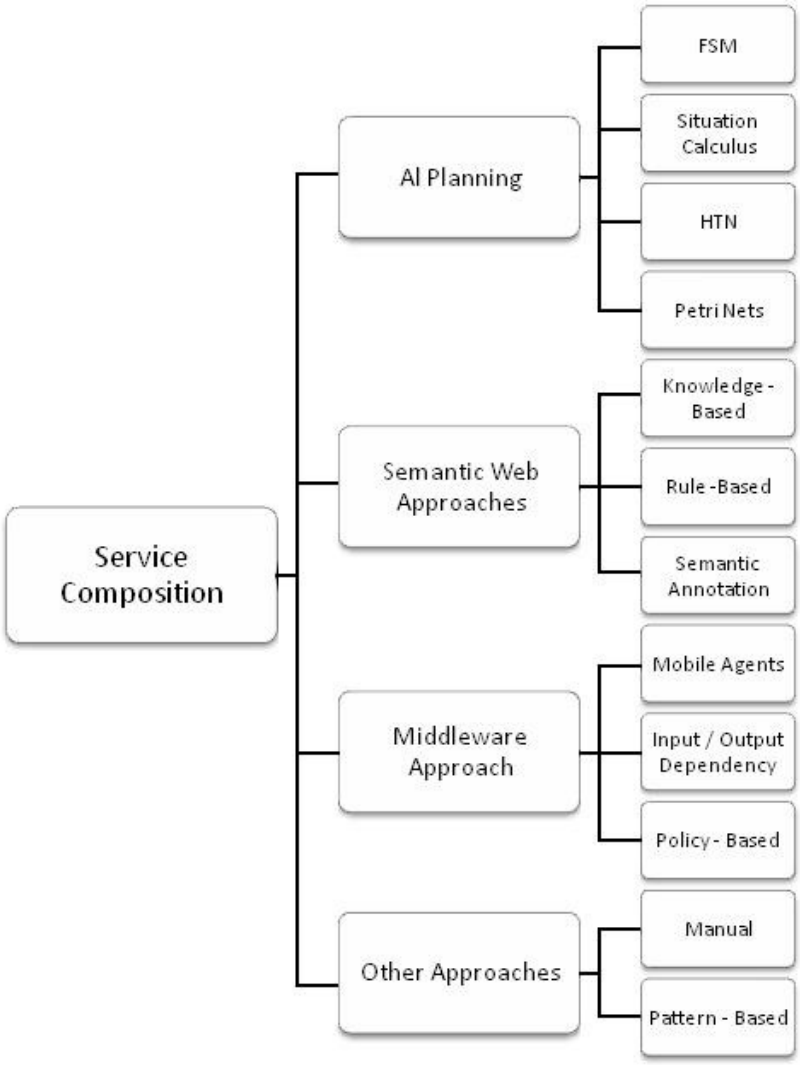

Figure 26 Service Composition Categorization

In general, there are several kinds of service composition. The traditional approach is called ***manual composition*** where users program and tell the system what to do during all the composition process development steps. Processes are defined using a process execution language like BPEL. Many plug-ins for tools like Net-Beans [44], JOpera [45] are available to enable manual composition.
The problem with such an approach is that it demands too much knowledge on the part of the user and it becomes more and more difficult with the explosion of web services resources.

The second approach is called ***automatic composition*** (without human involvement). It is used when the requestor has no process model but has a set of constraints and preferences. It is based on finding services for executing predefined abstract processes. The tools try to discover the available web services that semantically match as much as possible the user's needs [46]. Several approaches for automatic service composition have been introduced [47], including solutions based on Hierarchical Task Network (HTN), Golog [48], Artificial Intelligence (AI) planning or Rule-Based planning [47, 49-50]. However, automatic composition is still viewed as a task of high complexity because of the rapid proliferation of available services to choose from and the composition result risks to differ from the user's original goal.

The third approach is called ***semiautomatic or interactive composition***. In this kind of composition, the system usually helps users to find, filter and integrate automatically the desired services by matching the users requests with the available services. Moreover, it enables end users to intervene continuously during the composition process. Some efforts like OWL-S [51], METEOR-S [52] use semantic description of web services to help improving the discovery and composition processes. We believe that the semantic of the provided services could be used by tools or systems to guide the user to limit the available choices and to define his preferences to finally reach the composition goal.

5.4.1 Automated Web Service Composition Methods

Therefore, building composite Web services with an automated or semiautomated tool is critical. To that end, several methods for this purpose have been proposed. In particular, most researches conducted fall in the realm of workflow composition or AI planning.

Web service composition can be seen as the construction of a process to attain a certain goal. This is a problem which has been investigated extensively by research in Artificial Intelligence (AI). A classical planning problem includes a description of the initial state of the world, a description of the desired goal, and a description of the possible actions which may be executed. We make an overview of AI approaches to planning that have been investigated for the purpose of Web service composition.

5.4.1.1 Situation calculus

Situation calculus adapt and extend the Golog language for automatic construction of Web services. Golog is a logic programming language built on top of the situation calculus. The general idea of this method is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition and inter-operation. The user's request (generic procedure) and constraints can be presented by the first-order language of the situation calculus (a logical language for reasoning about action and change).

In Situation Calculus a dynamic world is modeled as progressing through a series of situations as a result of various actions being performed within the world. A situation represents a history of action occurrences. The constant 0 *S* describes the initial situation where no actions have occurred yet. The transition to the successor situation of *s* as result of an action *a* is denoted using *do* (*a*, *s*). Lastly, the truth value of statements given a situation is modeled by fluents which are denoted by predicate symbols and take a situation as their last argument. In [38] the authors argue that composition of Web services, viewed as execution trees, can be realized, at least under certain assumptions, in reasoning about actions formalisms, making specific use of Situation Calculus. In [39] the use of an augmented version of Golog is suggested. Golog is a high-level logic programming language, built on top of the Situation Calculus, for the specification and execution of complex actions in dynamical domains. The authors extended Golog to provide knowledge and sensing actions in order to become a suitable formalism for representing and reasoning about the semantic Web services composition problem. The general idea of the method described is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition and inter-operation.

5.4.1.2 Hierarchical Task Networks

Hierarchical Task Network (HTN) planning is an artificial intelligence (AI) planning method.

Hierarchical Task Network (HTN) [40] planning is a method of planning by task decomposition. In contrary to the other concepts of planning, the central concept of HTNs is not states, but tasks. An HTN based planning system decomposes iteratively the desired task into a set of sub-tasks until the resulting set of tasks consists only of atomic (primitive) tasks, which can be executed directly by invoking some atomic operations. During each iteration of task decomposition, it is tested whether certain given conditions are violated (e.g. exceeding a certain amount of resources) or not. The planning problem is successfully solved, if the desired complex task is decomposed into a set of primitive tasks without violating any of the given conditions. An approach of using HTN planning in the realm of Web Services was proposed in [41], facilitating the planning system SHOP2, which uses DAML-S Web service descriptions for automatic service composition.

Each state of the world is represented by a set of atoms, and each action corresponds to a deterministic state transition. However, HTN planners differ from classical AI planners in what they plan for, and how they plan for it. The objective of an HTN planner is to produce a sequence of actions that perform some activity or task. The description of a planning domain includes a set of operators similar to those of classical planning, and also a set of methods describing how to decompose a task into sub-tasks (smaller tasks). Given a planning domain, the description of a planning problem will contain an initial state like that of classical planning—but instead of a goal formula; the problem specification will contain a partially ordered set of tasks to accomplish. Planning proceeds by using the methods to decompose tasks recursively into smaller and smaller sub-tasks, until the planner reaches primitive tasks that can be performed directly using the planning operators. For each non primitive task, the planner chooses an applicable method, instantiates it to decompose the task into sub-tasks, and then chooses and instantiates methods to decompose the sub-tasks even further. If the plan later turns out to be infeasible, the planning system will need to backtrack and try other methods. [69]

SIRIN ET AL. presents in [66] a prototypical implementation of a service composer which bases on the HTN-planner SHOP2 [69]. The prototype transforms atomic services described in OWL-S into primitive HTN-tasks and composed services also described in OWL-S into non primitive HTN-tasks. As user request an initial state and a partially ordered set of services has to be provided instead of the goal of the user. This means the user has to compose the needed services on a high level on his own. If the services in the user request are assembled out of other services then they are decomposed by the prototype into atomic services. This decomposition will be optimised according to a given optimization rule in case several different decompositions are possible. This approach is more powerful than the Meteor-S prototype, since it can handle a replacement of two atomic activities by a superseding atomic activity. However like the Meteor-S approach the prototype presented by SIRIN ET AL. is very limited according its composition power since it heavily relies on given human modeled service compositions. But this limitation in power has naturally a positive effect on composition complexity, making HTN-planning relatively fast in comparison to the forthcoming approaches.

In the SHOP2 planner is applied for automatic composition of Web services, which are provided with DAML-S descriptions. SHOP2 is a Hierarchical Task Network (HTN) planner. The concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in DAML-S process ontology. The HTN planner is more efficient than other planning language, such as Golog.

SHOP2 is a domain-independent HTN planning system, which won one of the top four awards out of the planners that competed in the 2002 International Planning Competition. HTN planning is an AI planning methodology that creates plans by task decomposition. HTN planners differ from classical AI planners in what they plan for, and how they plan for it. The objective of an HTN planner is to produce a sequence of actions that perform some activity or task. The description of a planning domain includes a set of operators similar to those of classical planning, and also a set of methods, each of which is a prescription for how to decompose a task into subtasks. Planning proceeds by using methods to decompose tasks recursively into smaller and smaller subtasks, until the planner reaches primitive tasks that can be performed directly using the planning operators.

5.4.2 Semi-automated Composition of Services

Composition of services is a step by step assembling of self-contained services to process with a specific capability. Each assembling step consist at least of the following tasks:

- Service Discovery is the task of finding a service. Since services can be provided by various providers all over the world, finding of services is a non-trivial task. The challenges in finding proper services are the potentially huge number of service repositories as well as of different languages, law spaces, ontologies and business models. Searching for services on the world scale is the worst case according the named challenges.
- Service Matchmaking denotes the task of selecting a proper service from a list of existing services for assembling. Due to the potentially huge amount of services a matchmaking mechanism should have a high precision in selecting a proper service. This is important to avoid overwhelming the modeler with unsuitable services. Furthermore, the variability of possible service capabilities is high due to the high complexity of the world and the various business scenarios. This also demands from a matchmaking mechanism a high recall, especially for rare and special services. High recall means, that ideally all matching services are found. In case capabilities of rare or special services are needed, high recall is important to ensure that at least one of the existing and matching services is retrieved.
- Data-/Control-flow Linkage defines the logical sequence of services and specifies the needed data transfers. Depending on the desired capability of the final service composition and the capabilities of the existing services the control-flow of the services may vary from a simple sequence to complex control structures including branches, parallelism and loops. Furthermore data-flow can be in the simplest case just an opaque propagation of the output of a service to the input of another service. However in most cases data-flow will imply transformation of data and data structures to overcome different representations of the same or similar, but matching concepts.

5.4.3 Comparing Service Composition Approaches

	Service Connectivity	Non-Functional Properties	Composition Correctness	Automatic Composition	Composition Scalability
BPEL	✔				Average
OWL-S	✔	✔			Average
Web Components	✔		✔		Low
Π – Calculus	✔		✔		Good
Petri Nets	✔		✔		Low
Model Checking / FSM	✔		✔	✔	N/A

Table 5-1 Comparing service composition approaches [114]

A composition approach is the core part of a composition method. Several evaluations of compositions approaches can be found in literature [115,114]. [114] compare these approaches using the criteria: 1. Service connectivity, 2. Non functional properties, 3. Composition correctness, 4. Automatic composition and 5. Composition scalability. [115] use the following criteria: 1. Connectivity and Non-functional Properties, 2. Composition Correctness, 3. Automatic Composition, 4. Composition Scalability, 5. Exception Handling and Compensations and 6. Tool Support. The criteria used by both evaluations are similar, Beek et al. extend the set of criteria used by [114] by adding exception handling and compensations and by including tool support.

All composition approaches offer connectivity, without connectivity if would be impossible to integrate a set of services. Connectivity is an absolute minimal requirement for a composition approach.

The ability to handle non-functional properties (or Quality of Service attributes) of services is desirable, for example to assess the speed or error probability of a proposed composition. Most composition approaches neglect specification of non-functional Quality of Service (QoS) properties such as security, dependability, costs, response time, reliability and scalability. Only OWL-S lets users define non functional properties, but that capability has yet to be fully specified [114].

When the correctness of a composition is verified, the service and composition specifications are used to assess whether the composition will behave as required under different circumstances. The more formal approaches (Web components, π-calculus, Petri nets and Model checking/FSM) all provide some support for verification tasks. BPEL and OWL-S do not provide any way to check on specification conformation .

Automated composition is a part of the semantic web vision. Some kind of intelligent composition engine processes the specifications of the services and the requirements of the composition to be designed. A composition is then generated without any human involvement.

The composition scalability column indicates whether the composition approach is suitable for larger compositions. When the number of services used increases and the complexity of the composition approach increases at a higher rate, the scalability of the method is low.

The approaches listed in Table 5-1 assume that all services are well-defined and there are unambiguous selection requirements and there is a clear, undisputed, goal about the performance. In fact, these approaches can be better denoted using the term *representational formalism* than using the term composition approach.

A composition approach as representational formalisms is not sufficient in the field of e-government. None of the composition approaches support the orientation and negotiation phase. In short, the evaluation approaches focus on easy to measure criteria, whereas other requirements are equally or even more important. As such there is a need for an evaluation approach that is able to deal with this more difficult to measure requirements.

5.4.4 Comparison of AI Planning Techniques

5.4.4.1 Evaluation Criteria

We start by presenting a set of criteria that will be used to evaluate the planning techniques and planners surveyed. We selected criteria with the aim of evaluating the capability of planning techniques for automatic Web service composition. Table 5-2 and Table 5-3 present a list of the properties and control structure criteria respectively, and the reason for their choice.

Properties Criteria	Reason
Domain-independence	Allows the solution of a broad range of problems.
Search Mechanism	Mechanism used to prune the search space.
Partial Observability	The real-world situation is not fully observable. The ability to reason based on incomplete information is important.
Domain Complexity	Measures the level of difficulty in solving composition problems in terms of time and computation steps.
Scalability	Provides the ability to solve large real world problems.
Extendibility	The assumption that planning goals are sets of final desired state is unrealistic especially in Web service compositions. Because it is a reactive system, infinite plans might exist that react to changes.
Applicability	Standards to which it can apply to (i.e. BPEL4WS or OWL-S).

Table 5-2 Properties

Control Structure Criteria	Reason
Composition constructs	Sequential, iteration, parallel and conditional
Non-determinism	An action executed in a state may lead to many different states. The ability to deal with nondeterminism will ensure the output state is the desired one.
Concurrency	Allows more than one services to execute at the same time
Plan monitoring	The plan monitor is important to ensure the plan is executed as expected.
Plan failure	If the plan as generated fails, it should be able to detect and recover such failures to retain correctness of composition.

Table 5-3 Control structure

5.4.4.2 Discussion

Based on the analysis conducted, we see that HTN planning is well suited for Web service composition. We also noted the following:

- OWL-S is most used when comes in applying AI planning to Web service composition.
- The domain complexity of estimated-regression planning, planning as Model Checking and MDPs hampers the scalability to large-scale problems. In Web service composition, the problem domain tends to very large. Therefore, these techniques are not feasible for large Web service composition.
- There is no existent planner based on estimated-regression and situation calculus that supports an extended goal in Web service composition. The ability to support an extended goal is important in automatic Web service composition, because requirements (the desire goal) set by users differ from time to time.
- Plan monitoring and plan failure detection and recovery are not supported by most planners. Automatic Web service composition should not only generate the plan automatically, but it should be able to monitor the plan during execution and perform recovery when failures occur.
- Estimated-regression planning [117] is suitable for Web service composition. However, there are too many unresolved issues, such as dealing with non-deterministic actions.
- Using pure HTN planning hampers the ability to tackle recursive and conjunctive tasks. Zhang [119] proposed an enhanced HTN planning algorithm to tackles these problems. The algorithm combines HTN methods with partial-order planning (POP) for Web service composition. HTN methods are used for automatic plan generation and POP is responsible for plan refinements.
- Recently, Kuter [118] found that the way ND-SHOP2 solves non-determinism is not efficient if search strategies cannot cut down the search space. Therefore, a novel algorithm, YoYo, is proposed to solve non-determinism under fully observability. The algorithm combines the power of search-control strategies in HTNs with Planning via Symbolic Model-Checking.

The result of the analysis shows that HTN planning has various advantages over other planning techniques that are currently used to automate the composition of Web services. However, the analysis conducted only considers the composition process itself, the discovery of services has not been taken into account.

Properties Criteria	Domain-independent heuristic (Optop)	HTN Planning (SHOP2)	Situation Calculus (Golog)	Planning as Model Checking [FSM] (ASTRO)	Planning based on Markov Decision Processes [Petri nets]
Domain-independence	Domain-independent	Domain-specific or domain-configurable	Domain-independent	Domain-specific	Domain-specific
Search Mechanism	Search guided by heuristic	Forward search	Based on current situation	Breadth-first search proceeding backward from the goal state toward the initial state.	Policy guided the search
Partial Observability	know-val predicate	WSC-SHOP2 is a sound and complete HTN-planning algorithm for solving planning problems with incomplete information about the initial world state.	Uses sensing actions during execution or knowledge	Yes	Original MDP does not support partial observability. POMDP is the partial observability version of MDP.
Domain Complexity	Building the regression-match graph is too expensive (no proper measurement exists)	O((n/k)!·k)	Plan generation takes linear time. Combination of complex actions is polynomial in the number of primitive action occurrences in its definition	O(n!)	$O(\|S\|^2)$
Scalability	Not feasible	Scales well to large domain problems.	Reasonable	Can deal with large scale problems. However, verification steps are time consuming.	Lack of scalability.
Extendibility	N/A	Define as HTN Methods	N/A	Yes, but have to take into account the context of execution (i.e. state of service)	Yes
Applicability	Mapping between OWL-S and PDDL	Translation between OWL-S and planning Operators	Translation: OWL-S to PDDL to situation calculus	Translation between BPEL4WS and planning operators	Converting a policy into a workflow in BPEL4WS specification using BPWS4J API

Table 5-4 Properties comparison of AI planning techniques and planners

Properties Criteria	Domain-independent heuristic (Optop)	HTN Planning (SHOP2)	Situation Calculus (Golog)	Planning as Model Checking [FSM] (ASTRO)	Planning based on Markov Decision Processes [Petri nets]
Composition constructs	Sequential and parallel	Sequential, parallel and conditional	Sequential, iteration and conditional	Sequential, iteration and conditional	Sequential, iteration and conditional
Non-determinism	Not addressed	Can be included during develop-ment of methods. ND-SHOP2 is specifically design to deal with non-determinism	Represent with disjunctive formulae	Though weak and strong (acyclic) graphs.	Yes
Concurrency	Via interleaving	Fully supported	Concurrency via interleaving (ConGolog – concurrency Golog)	Supported	Concurrent MDPs (extension to traditional MDPs)
Plan monitoring	Not addressed	Debugging	Not addressed	Supported (WS-mon)	Not addressed
Plan failure detection and recovery	Not addressed	Backtracking mechanism, precondition reasoning, post-condition check	Not addressed	Recovery mechanism is not addressed	Not addressed

Table 5-5 Control structure comparison of AI planning techniques and planner

6 Policy based management

6.1 Introduction

This section focuses on the paradigm of policy-based management (PBM) by providing an overview of the state-of-the-art and an elaboration on the mapping to pSHIELD especially to its scenario. The motivations behind the interest in PBM are clarified besides a description of what is meant by policy (its types) and PBM. The typical architecture followed in PBM will be detailed with an emphasis on policy specification means, their discussion and affiliated protocols.

6.1.1 Policy

A policy can be abstractly seen as a mapping from a set of conditions to a set of actions. Also, policies are meant to serve as the governing reference for any required adaptation a system may require. Others see a policy as in: “information which can be used to modify the behaviour of a system”(Lupu, et al., 1999). While in (Verlaenen, et al., 2007) policies are seen as a means to “configure the behaviour of services” and are described in a declarative high-level manner that identifies what should be done in a specific situation but not how it should be done. All of the above descriptions converge to the same essence that a policy is intuitively a set of rules drawn in a high-level manner where under a given evaluated situation; certain actions are dispatched for execution.

6.1.2 Policy-Based Management

Management comes as an intrinsic requirement for virtually all systems and indeed varies in type. Essentially, Policy-Based Management (PBM) can be seen as a type of system management that allows for minimal manual intervention in controlling, (re)configuring and monitoring system’s constituents and its overall behaviour through predefined governing policies. In (Lymberopoulos, et al., 2003), the objective of PBM is identified as to “allow flexible and adaptive management where the policies define the adaptation choices or strategy which can be modified without recoding or even shutting down the system.” This objective stresses on that PBM is meant mainly to provide for and regulate the adaptation a system may require at certain points throughout its lifetime. However, PBM is not meant solely for adaptation but also to aid in configuring, controlling and monitoring different system constituents (or components). Moreover, PBM allows for system adaptation without the need for changing its implementation (Verlaenen, et al., 2007). A specialisation of PBM that emphasise on managing networks and their services from a business perspective through the usage of policies is referred to as Policy-Based Network Management (PBNM) (Strassner, 2003). We mention the latter for disambiguation reasons but we continue to concentrate on PBM in this section as it encapsulates the essentials of PBNM.

6.1.3 Motivations

The rationale behind choosing PBM comes as a natural consequence of the advancement of computerised systems in general. Such systems tend to increasingly be distributed, complex, and heterogeneous operating in a dynamic surrounding environment. Assuming the efficiency of human intervention and continuous management is not realistic especially if the system is bound to alter its behaviour frequently and adapt to emerging situations. Consequently, PBM comes -and increasingly so- as an adopted simplifying approach to handling various operational, administrative and configuration matters in different computerised systems (Verma, 2002). Essentially, allowing designers to specify in a high-level descriptive manner a system’s policy is advantageous especially as they are consequently relieved from defining various and often heterogeneous lower level strategies. These strategies are inferred by the PBM which is responsible for interpreting the required steps and actions to be distributed and enforced system-wide. More importantly, it will minimize human intervention in case of system adaptation and will ensure the avoidance of conflicts among the policies to enforce. However, the use of PBM does rely heavily on the quality of identified policies and hence they need to be carefully drawn.

6.2 Typical Architecture

A typical PBM architecture is defined by the IETF policy framework (IETF, 2000). The architecture constitutes several points and elements, i.e., Policy Management Tool (including the required tools and a policy repository), Policy Decision Point and a Policy Enforcement Point. Policy specification is also considered as a main element and is discussed separately from the architecture onwards, see Section Policy Specification.

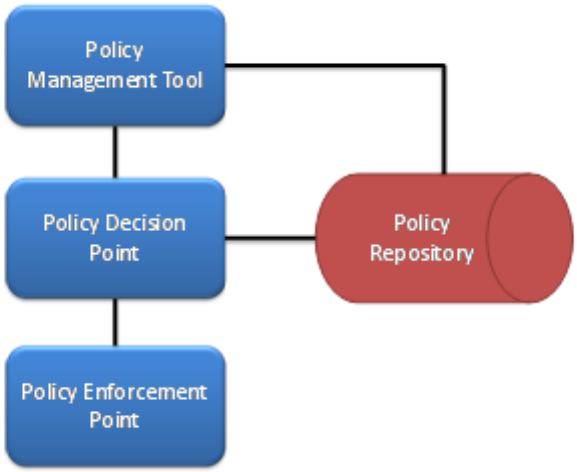

Figure 27 Typical IETF PBM Architecture

Figure 27 presents an illustration of the main points in a typical PBM.

6.2.1 Policy Management Tool

Different terminologies are used to refer to PMT such as Policy Administration Point (PAP) for instance. PMT is mainly used by the administrator(s) in order to specify business-level (high-level) abstractions that constitute polices. A number of elements are needed at this point typically (Verma, 2002; IETF, 2000):

1. A user interface that could be graphical with command-line support. Used as a policy editor with simple validation
2. A resource discovery element that determines the network topology, its capabilities, constituents, users and running applications
3. A policy translation (or transformation) element that transforms high-level policies into a lower-level constituents-specific policies. It also ensures policies' consistency, correctness and distribution feasibility through a validation process
4. A policy distributer/storage-retrieval element; as the name suggests, it interacts with the policy repository (explained onwards) to store low-level policies and allow for their retrieval

An example of policy translation as described in (IETF, 2000) would be; assume a high-level policy segment that defines *"Premium Traffic between Point A and Point B"*. This can be translated into a low-level policy rule: *"source = 10.24.195.x, dest = 10.101.227.x, any protocol, perform Premium Service action".* Indeed, a validation check can only be carried out in an offline manner here where the syntactic and semantic integrity of each policy must be preserved. Some semantic validation checks are defined by (Verma, 2002) as in:

1. Bounds checks: ensure that a given attribute value is within a predefined range
2. Relation checks: ensure that any two values assigned to interrelated policy parameters are satisfactory to their relationship
3. Consistency checks: ensure a conflict-free set of policies
4. Dominance checks: ensure that all specified policies are reachable and are to be active at some point during the system's lifetime

5. Feasibility checks: these are domain dependent checks that need to ensure that the underlying environment can support the specified policies

Moreover, while consistency checks need to ensure that conflicts among policy rules are avoided and given that this is an offline stage, other checks should be carried out at runtime to avoid potentially triggered conflicts.

6.2.1.1 Policy Repository

A policy repository is mainly concerned with managing translated policies, e.g., Directory Server, Database. It should allow for the storage, search and retrieval of policies and interface with other elements using for instance, a Lightweight Directory Access Protocol (LDAP) protocol (see Section LDAP).

6.2.2 Policy Decision Point

PDP is mainly a set of modules that are capable of examining applicable policies and consequently determine the decisions required for the system to comply with that policy. PDP is responsible for communicating policy-inferred decisions/actions to the Policy Enforcement Point (PEP) that could reside on several physical devices. That channel of communication is governed by a protocol such as SNMP (see Section SNMP). PDP also needs to interact, (i.e., fetch policies) with the policy repository using a protocol such as LDAP.

6.2.3 Policy Enforcement Point

PEP is the final point in a typical IETF policy framework architecture. PEPs act as logical entities that interface between systems' devices/resources such as sensors, where they are likely to reside, and the PDP by processing exchanged requests and responses. As the name suggests, PEP is responsible for enforcing actions communicated from the PDP at the device-level. Those actions reflect the policy or policies to be deployed at the local level.

6.2.4 Policy Types

Policies can naturally be divided into two main types, i.e., functional and on-functional policies. Functional policies are intrinsic to the system's operational tasks which is more of an application-specific issue. On the other hand, non-functional policies for instance, those meant for maintaining a certain level of fault tolerance, security and Quality of Service (QoS), tend to be less dependent on the application nature but not completely (Robben, et al., 1999).
An adapted example inspired from (Matthys, et al., 2008) illustrates how a high-level security policy is mapped to a more detailed policy for door monitoring data, see Figure 28. Detailed or lower-level policies may sometimes comprise functional policies such as enabling/disabling access control on a gateway.

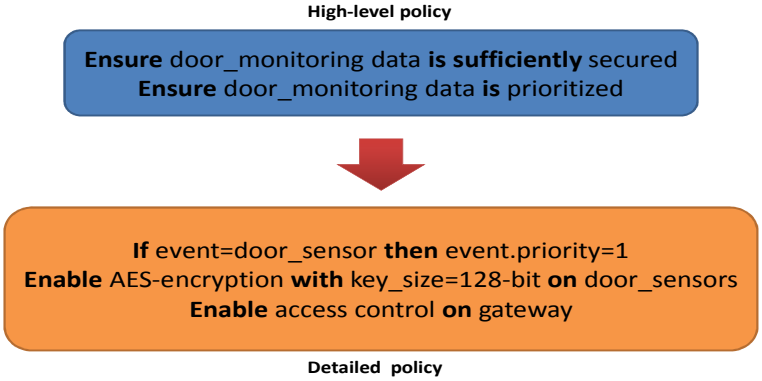

Figure 28 Non-Functional Policy Transformation Example (Matthys, et al., 2008)

In the aforementioned example, the need to prioritize door monitoring data results in assigning a high priority to the data originating from that door (by dedicated sensors) and that is encapsulated in a

door_sensor event. Moreover, in order to ensure sufficient security is applied on the exchanged door_monitoring data, the translation to lower-level policy specifies the encryption algorithm to be used with the adequate key size proportional to the security level requested in the high-level policy.
From a network perspective, policy-based QoS helps manage network traffic through regulating and controlling bandwidth cost (with possible negotiation with bandwidth providers) which results in a better end-user experience (Microsoft, 2009). For example, QoS policies can specify Differentiated Services Code Point (DSCP) (Cisco, 2005) values and throttling rates on routers and switches in order to balance the cost of service and the network performance. DSCP values are used to classify traffic into different levels of importance by assigning priority values to different packets, (e.g., high priority, best effort, low priority) where throttling can be used to set the rate of outbound traffic.

6.3 Policy Specification

A representative group of state-of-the-art policy specification languages/models are presented here. Policy specification is essentially the initial phase where researched policies are concretized using a model or language of choice. The latter can be regarded as an attempt to formalize administrators' intents in a machine understandable form.

6.3.1 XACML

The eXtensible Access Control Markup Language (XACML) is an XML-based declarative language designed for specifying access control policies and follows a specific processing model. Up to date, the most recent version is XACML 3.0 that was ratified by OASIS (OASIS) and includes several additional features such as delegation.

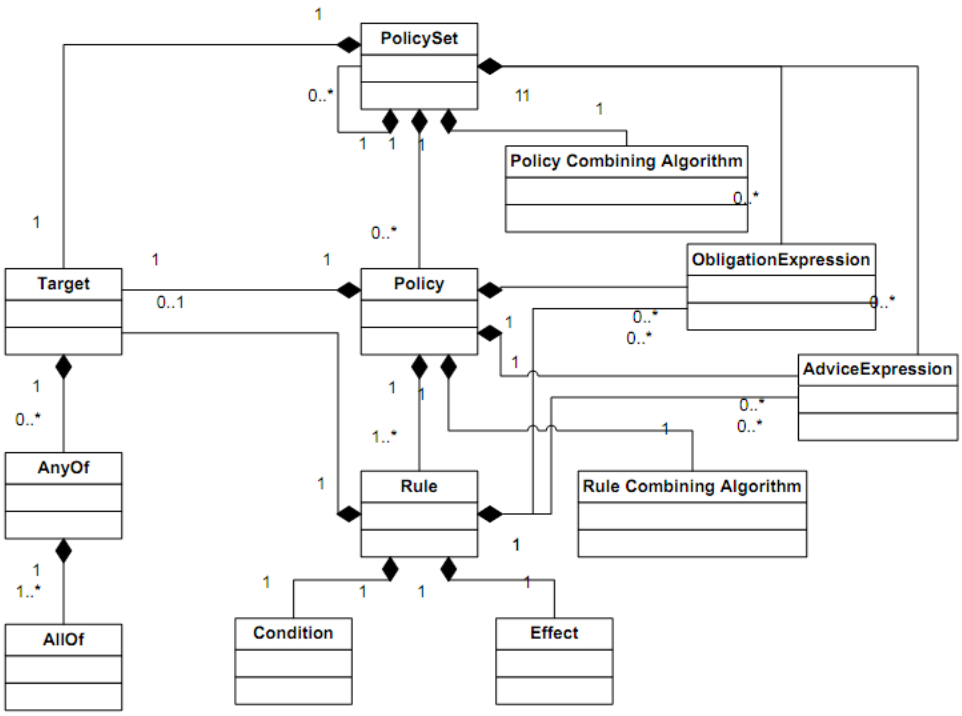

Figure 29 XACML Policy Language Model (OASIS, 2009)

XACML comprises a large set of abstracted elements. A comprehensive list of those elements is presented in Section 5 of (OASIS, 2009). The main policy language model used in XACML is presented in Figure 29 where it exhibits three essential elements/components, i.e., the *policy*, the *policySet* and the *rule*. The *rule* is the most basic constituent of the *policy* that usually encapsulates a number of rules. A given *rule* is evaluated based on its composing elements that are mainly: *target*, *effect,* (i.e., Permit/Deny), *condition,* (i.e., Boolean expression - optional), *obligation* and *advice*. Besides holding a set of rules, a policy comprises other elements such as a *target*, *rule-combining algorithm identifier*, *obligations* and *advice*. This is similar to the *policySet* in terms of constituents but with the differentiation that a *policySet* holds a set of policies and *policy-combining algorithm identifier* whereas a *policy* holds a set of *rules* and *rule-combining algorithm identifier*. Therefore, *target*, *obligations* and *advices* are relevant to policies in a given *policySet* while they are relevant to *rules* under a given *policy*. Essentially, an XACML *policySet*, *policy* or *rule* element comprises a *target* that specifies the set of requests to which it applies. For example, a *rule target* specifies the set of requests in the form of a logical expression on some or all attributes in the request. Also, a rule/policy-combining algorithm specifies the procedure through which an authorization decision can be reached based on the evaluation results of a set of rules/policies.

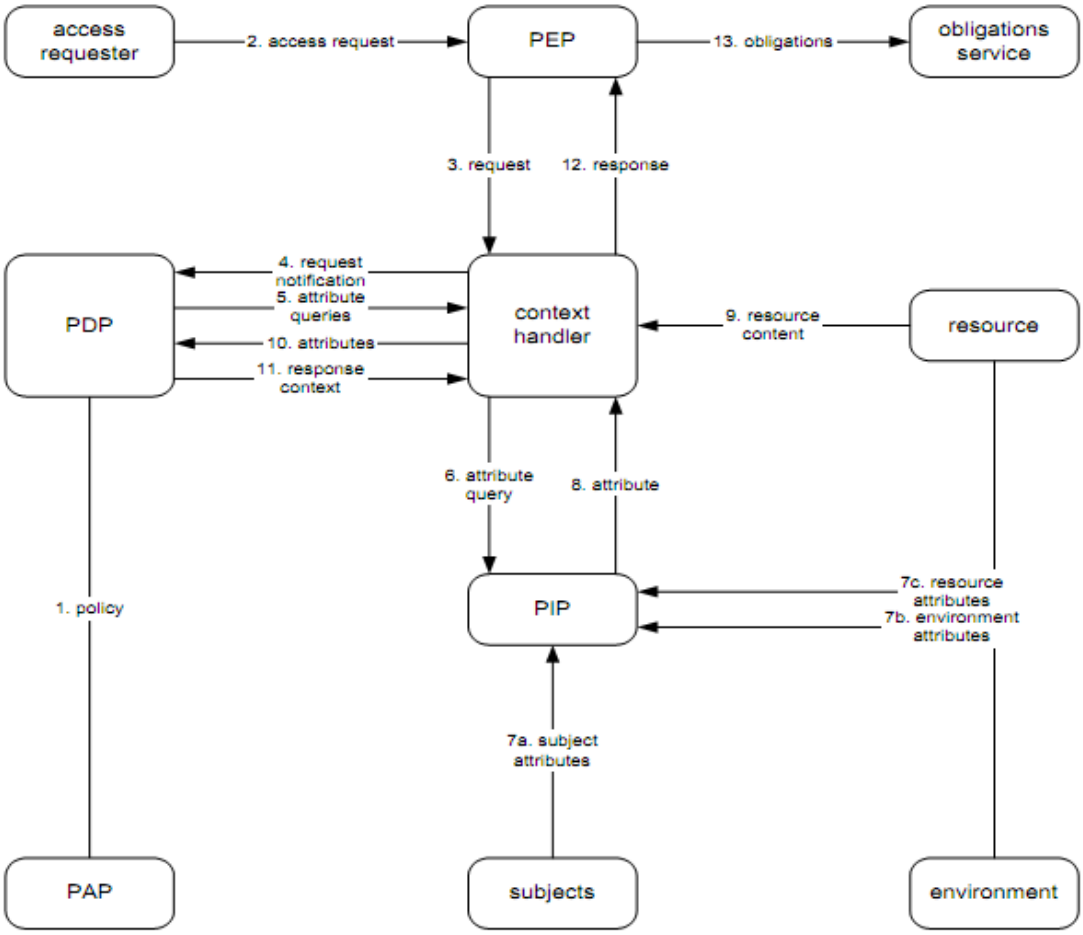

Figure 30 XACML Data-Flow Diagram (OASIS, 2009)

Referring to the XACML data-flow diagram (see Figure 30) clarifies more the connections among the different policy model elements. The architecture can also be seen to share some of the PBM typical architecture defined by the IETF that was discussed earlier. The following is a description based on (OASIS, 2009) of the entities mentioned in the XACML data-flow diagram:

- Policy Enforcement Point (PEP): perform access control by issuing decision requests and enforcing authorization decisions
- Policy Information Point (PIP): act as a source of attribute values
- Policy Administration Point (PAP): responsible for creating a policies or policy sets
- Policy Decision Point (PDP): evaluate the applicable policy and renders an authorization decision
- Context Handler:
    - Convert decision requests in the native request format to the XACML form
    - Convert XACML authorization decisions to the native response format
- Environment: “The set of attributes that are relevant to an authorization decision and are independent of a particular subject, resource or action”
- Resource: “Data, service or system component”
- Subject: “An actor whose attributes may be referenced by a predicate”
- Obligation: “An operation specified in a rule, policy or policy set that should be performed by the PEP in conjunction with the enforcement of an authorization decision”

PAP composes policies and policy sets which are made available to the PDP. The communication between the PAP and the PDP can be facilitated by a repository that XACML does not impose restrictions on its location. This can also apply to the communication between the context handler and the PIP. In a typical scenario, the access requester sends its request to the PEP which forwards it (possibly including attributes of the subjects, resource, action, environment, etc.) to the context handler. The latter converts

the native request to an XACML format and sends it to the PDP that could request additional attributes from the context handler itself. The context handler retrieves the needed attributes through the PIP and returns them to the PDP that evaluates the policy. Consequently, the PDP returns the authorization decision to the context handler which converts it to the native response format and sends it to the PEP. Finally, the PEP fulfils any obligations if present and permits or denies access to the resource.
An XACML implementation (up to version 2.0) made by SUN is available online under a reasonably flexible license (See (Sun, 2004)). Moreover, a comprehensive presentation detailing many aspects of XACML with elaborating examples is available from *Axiomatics AB* in (Gebel, 2010).

6.3.2 IBM EPAL

Enterprise Privacy Authorization Language (EPAL) (IBM, 2003) is an IBM developed (proprietary) policy specification language that is a subset of XACML and mainly aimed at the support of enterprise privacy policies. Where EPAL is close to XACML's structure and concept it however falls short in many aspects against XACML. In (Anderson, 2006), a detailed comparison between EPAL v1.2 and XACML v2.0 is presented which clearly shows how XACML is a more comprehensive and standardized access control and privacy policy specification language. For example, EPAL does not support nested and distributed policies besides the lack of digitally signed policies. Moreover, EPAL allows only one subject per access request and evaluates only first applicable rule. All of this is added to EPAL's inconsistent treatment of attributes and limitations when it comes to (hierarchical) roles. Indeed, XACML especially v3.0 provides a more complete privacy and access control policy specification solution than EPAL (Anderson, 2006).

6.3.3 WSPL

Web Services Policy Language (WSPL) (Anderson, 2004) is an XACML-based policy specification language that focuses on the web services domain. WSPL is driven by the intention of standardization. It is motivated by the several aspects the web services domain comprises that can be controlled and described using policy rules. Those aspects could involve authorization, authentication, quality of protection and service, messaging reliability, privacy and other service-specific options. A main feature in WSPL is policy negotiation and its ability to merge different policies which represent the intersection of such policies if such an intersection resulting policy is valid (see Figure 31 for an elaboration). Policies in WSPL can be also based on parameter (standard data-types and functions) comparison as opposed to pure equality matching. This allows for a more "fine-grained" choice of parameters (Anderson, 2004).

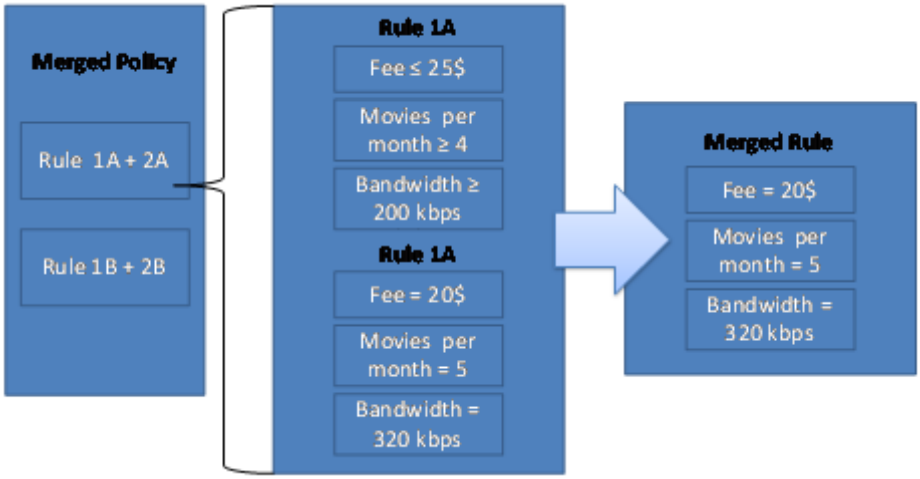

Figure 31 An Example of Policy Merging in WSPL (Anderson, 2004)

WSPL was mainly developed based on a number of use cases that were analysed and reviewed for that purpose in a public forum. Indeed, it was henceforth formally analysed and is seen to be "good" basis for a web services policy standard.

6.3.4 Ponder (2)

Ponder (Damianou, et al., 2001) is an Object-Oriented declarative policy specification language aimed at defining management and security policies for distributed systems. The language was developed at the Imperial College London over several years. Ponder includes essential interrelated concepts in its core, i.e., domains, roles, relationships and management structures. Domains are used to group objects of similarly applicable policies where roles are used to group policies for identified positions in a given organisation. Relationships are used to specify interactions between roles where a management structure is used to capture the configuration of roles and relationships for an organisational entity such a business unit.
The base-class diagram used in Ponder is presented in Figure 32. The detailed explanation of the diagram is out of scope but a comprehensive description can be found in (Damianou, et al., 2000).

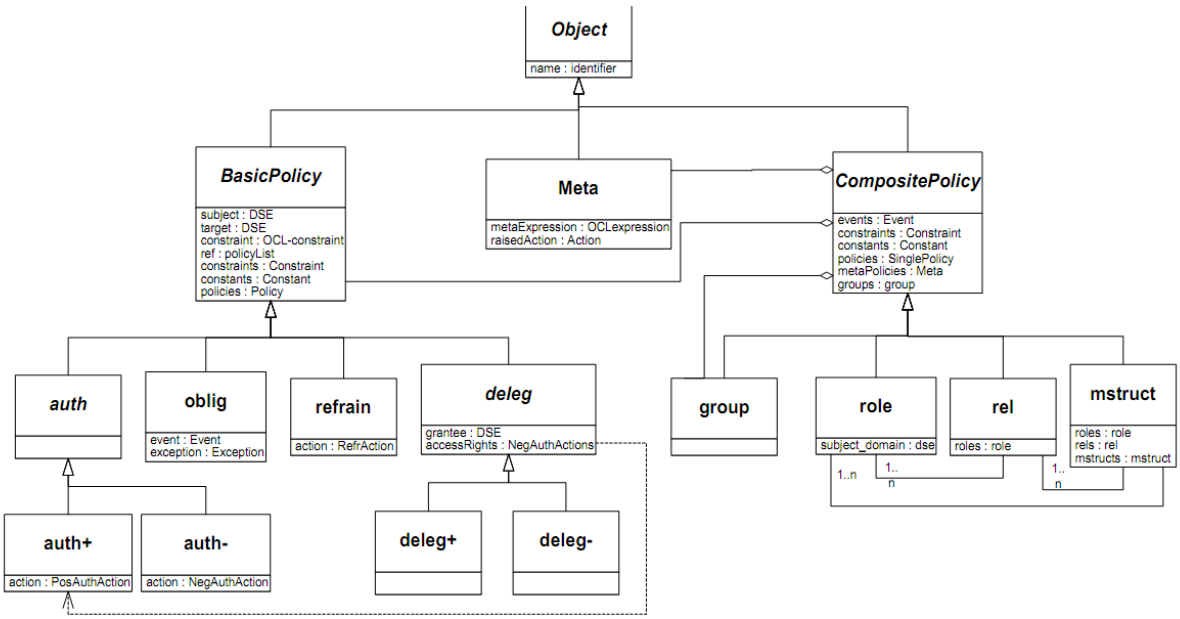

Figure 32 Ponder Base-Class Diagram (Damianou, et al., 2000)

Moreover, authorisation polices in Ponder can be implemented using available control methods such as those in firewalls and operating systems. Also, obligation policies (event-based condition-action rules) are supported. In addition, Ponder can be used for security management in distributed systems thus allowing for user registration and activity logging.
Onwards, two simple examples of authorization and role polices are presented (please refer to (Damianou, et al., 2000) for more examples). Figure 33**Errore. L'origine riferimento non è stata trovata.** presents the authorisation policy example where members of the *NetworkAdmin* domain are allowed to load, remove, enable or disable objects of type *PolicyT* in the *Nregion/switches* domain.

```
inst auth+ switchPolicyOps {
        subject          /NetworkAdmin;
        target     <PolicyT> /Nregion/switches;
        action     load(),    remove(),    enable(),
disable() ;
}
```

Figure 33 Ponder Authorisation Policy Example (Damianou, et al., 2001)

In Figure 34, a Ponder role policy example is presented. A role type is modelled for a telecommunication service engineer. In this role, the service engineer is responsible for handling customer complaints in addition to service requests. The role takes as input the calls database that provides information about subscribers' calls. In the case of a *customerComplaint* event, the *serviceComplaint* obligation is triggered where the subject of the role follow a sequence of actions on the calls database. All policies within this role have the same implicit subject so the obligation policy does not need to specify a subject explicitly.

```
type role ServiceEngineer (CallsDB callsDb) {
        inst oblig serviceComplaint {
                on          customerComplaint(mobileNo) ;
                   do       t.checkSubscriberInfo(mobileNo, userid) ->
                            t.checkPhoneCallList(mobileNo) ->
                            investigate_complaint(userId);
                target      t = callsDb ; // calls register }
        inst oblig deactivateAccount {...}
        inst auth+ serviceActionsAuth { . . . }
        // other policies
}
```

Figure 34 Ponder Role Policy Example (Damianou, et al., 2001)

While having been successfully used in several applications, Ponder suffered from different disadvantages which triggered a more recent and revised Ponder2 (Twidle, et al., 2009). Ponder2 aims at pervasive systems where Ponder's limitations such as the centralised infrastructure design, lack of support for collaboration between different polices' elements and the off-line nature of its policy specification task, are all dealt with. Hence, Ponder2 was implemented as a self-managed cell: "A self-managed cell is defined as a set of hardware and software components forming an administrative domain that is able to function autonomously and is capable of self-management" (Twidle, et al., 2009). Ponder2 allows for user-written Model Objects using Java with simple annotations. A smalltalk-like language, namely, PonderTalk is used for message communication among different Message Objects. Ponder2 is available for download under the GNU Lesser General Public License as published by the Free Software Foundation.

6.3.5 IBM TPL

The Trust Policy Language (TPL) (Herzberg, et al., 2000) was developed by IBM with an XML-based syntax. The language was only meant to provide for access control to resources. Its policy rule includes a set of certificates that are required for a given user to belong to a given group and an evaluation function to decide on that. That function uses fields from the included certificates in the evaluation process. TPL is limited only to defining access control for security policies. However, TPL allows policy authors to express constraints on several levels such as on (combinations) of credentials/inter-credentials and authentication. TPL has a definite version referred to as DTL where negative rules are not allowed. This allows for DTL to be easily mapped to a logic programming language. Based on (Seamons, et al., 2002), which provided a set of requirements to evaluate different policy languages for trust negotiation, TPL was found to be fulfilling some of the requirements. However, Portfolio and Service Protection *Language* (PSPL) was found to be a better policy language than TPL for trust negotiation.

6.3.6 PeerTrust

PeerTrust (Nejdl, et al., 2004) is a policy management framework based on the first order logic of Horn rules. It is mainly aimed at specifying requirements for security and trust on the sematic web. An elaborating example is provided in Figure 35 which forms a part of a more detailed informative scenario in (Nejdl, et al., 2004).

```
E-Learn:
discountEnroll(Course, Party) $ Requester = Party←
        discountEnroll(Course, Party).
discountEnroll(Course, Party)←
        eligibleForDiscount(Party, Course).
eligibleForDiscount(X, Course) ←preferred(X) @ "ELENA".

preferred(X) @ "ELENA"←
        signedBy ["ELENA"]
 student(X) @ "UIUC".
```

Figure 35 PeerTrust Policy Example (Nejdl, et al., 2004)

In Figure 35, an institute called E-Learn sells learning resources and provides discounts to some users. Discount eligible users are classified as the preferred customers in the ELENA group. Given that E-Learn was given a singed rule by ELENA specifying how to check whether a user is a preferred one or not, E-Learn does not need to communicate with ELENA each time it needs to check for discount eligibility.
Update of PeerTrust was halted 2004-2005 and was followed by a different project called Protune which is covered in the section to follow.

6.3.7 Protune

The PRovisional TrUst NEgotiation (Protune) (Bonatti, et al., 2005) is a framework for policy specification and deployment that merges between distributed trust management policies, business rules and actions required for access control. Its policy language is based on two existing languages, namely, PAPL (concerned with trust negotiation) (Bonatti, et al., 2000) and PeerTrust. Protune also provides a meta-language for handling critical negotiation decisions as well as integrity constraints that are meant for monitoring the disclosure of credentials. Protune follows an ontology-based approach for easier importing and exporting of metapolicies and also to allow for further language extension. The Protune project is maintained online at (Pro11) where an implementation is available for download including a language editor.

6.3.8 KAoS

KAoS (Uszok, et al., 2004) is a framework for policy and domain services that is organised in a set of components compatible with several agent platforms such as DARPA's CoABS Grid and Cougaar agent framework and CORBA. KAoS makes essential use of OWL which is an ontology authoring standard based on descriptive logic. OWL is used in KAoS to represent policies and domains besides other managed entities and affiliated elements. Also, OWL allows for the framework to be easily extended while being consistent with advanced semantic requirements.
KAoS domain services allow for semantically describing and structuring of software components, people and resources into groups of (sub)-domains which is believed to ease external policy administration and collaboration. Moreover, KAoS policy services are used to specify, manage and enforce policies as well as resolve conflicts among policies.

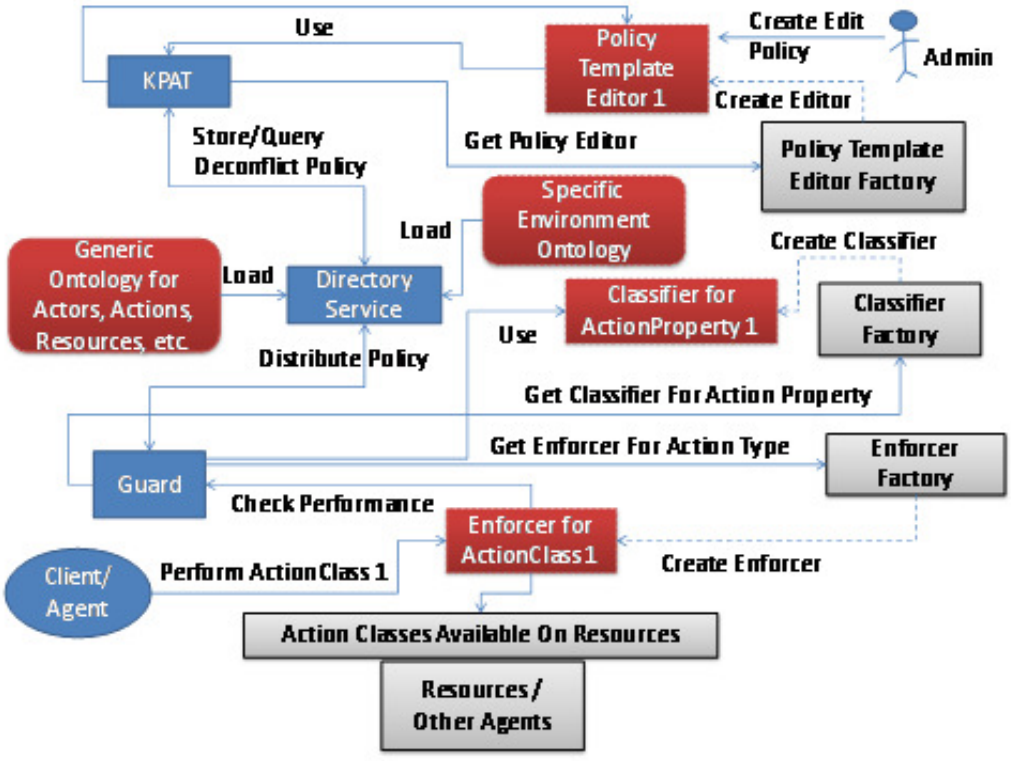

Figure 36 Basic KAoS Framework Elements (Uszok, et al., 2004)

Three types of conflicts can be detected in KAoS that are based on the positive or negative nature of an authorisation or obligation:

- positive vs. negative authorisation
- positive vs. negative obligation
- positive obligation vs. negative authorisation

KAoS resolves these conflicts through "harmonization" which involves assigning different policies different priorities (Bonatti, et al., 2007).
In Figure 36, the basic elements of the KAoS framework are presented. Two groups of functionalities are naturally found in the framework, i.e., generic and application-specific. Generic functionalities include ontologies creation and management as well as policies storage, querying, distribution, enforcement, disclosing and conflict resolving. Moreover, application-specific functionalities may include defining new ontologies and creating extension plug-ins such as policy templates, custom action property editors and subclasses action enforcers and classifiers.
More details can be found on the project's website at (KAoS) where the policy ontology used is also presented in details.

6.3.9 REI

REI (Kagal, 2002) is a policy language similar to what is used in the KAoS framework in many forms. REI was designed to deal with security issues where heterogeneous entities exist in a dynamic environment through policies. REI uses OWL-Lite and logic-like variables which allow for easier application-specific extensions and different relations specification. Polices here are defined based on what entities can/cannot do or should or should not do. REI policies follow an ontology that includes different concepts such as permissions, obligations, actions etc. as well as the ability to import domain/application-dependent ontologies. Analogous to KAoS's positive/negative authorisations and obligations, REI policies include "deontic concepts" such permissions/prohibitions and obligations/dispensations. Moreover, REI allows for rights and obligations to be dynamically exchanged among entities through right delegation and revocation as well as action or delegation request and cancelation. Policy conflict detection and resolution is handled through overriding policies which is a similar technique as in prioritisation in KAoS.
The project ended in May 2005 but its website is still alive at (REI, 2005) where more details can be sought if needed.

6.3.10 Discussion

Several reviews have already discussed and compared the aforementioned policy specification languages and models. Based on the following studies (Olmedilla, 2008)(Duma, et al., 2007), a set of metrics is selected to form a representative criteria for evaluating these policy languages/models in light of pSHIELD's characteristics. Following is a description per each:

- **Well-Defined Semantics:** the meaning of the policy is independent from the implementation of the language it is written in
- **Access Control:** requesters with valid credentials are given access to certain data/attributes
- **Minimal Information Disclosure:** credentials or attributes sensitivity specification
- **Mutual Exclusiveness:** control of simultaneous release of data that could be sensitive collectively
- **Sensitive Policies Protection:** control the disclosure of policies and assigning them sensitivities
- **Usage Control:** ability to impose restrictions and obligations on released data consumers
- **Action Execution:** allowing policy writers to specify actions in policies
- **Delegation:** passing on or transfer rights such as access rights
- **Evaluation Type:** distributed (able to gather policies or policy elements spread on the net for evaluation) or local policy evaluation
- **Evidences:** support for the concept of user credentials
- **Extensibility:** ability to adapt to user needs
- **Standardization:** the extent to which the approach has been standardized

Criterion	XACML v.3	EPAL	WSPL	Ponder	TPL	PeerTrust	Protune	KAoS	REI
Well-Defined Semantics	N	Y	N	N	N	Y	Y	Y	Y
Access Control	Y	Y	Y	Y	Y	Y	Y	Y	Y
Minimal Information Disclosure	Y	Y	-	Y	-	-	Y	-	N
Mutual Exclusiveness	-	-	-	Y	-	-	Y	-	Y
Sensitive Policies Protection	Y	-	-	Y	-	-	Y	-	Y
Usage Control	E	-	-	Y	-	-	N	-	Y
Action Execution	Y	Y	Y	Y	N	Y	Y	N	N
Delegation	Y	N	N	Y	N	Y	Y	N	Y
Evaluation Type	Dist.	L	Dist.	L	L	Dist.	Dist.	L	Dist.
Evidences	E	N	N	-	Y	Y	Y	N	-
Extensibility	Y	Y	N	Y	N	Y	Y	Y	Y
Standardization	A	U	-	-	-	-	-	-	-
A: Available, U:Unmaintained, Dist.: Distributed, L: Local, E: Extendable, -: Not Available/Applicable									

Table 6-1 Policy Language/Model Evaluation Table

Table 6-1 presents a comparison among the described policy languages/models based on the selected criteria. Support of privacy-centric metrics such as minimal information disclosure or usage control could not be verified for policy languages such as KAoS, EPAL, PeerTrust, etc. It can be noticed that all described policy languages support access control while they vary in their support to other metrics. XACML, Ponder, Protune and Rei seem to satisfy most of the metrics with the exception of well-defined semantics for XACML. However, XACML is the only model that is well standardised and maintained which makes it a viable model for adoption. EPAL had some standardization efforts by W3C in 2003 but it has not been maintained where XACML has been identified as better option in an evaluation against EPAL (Anderson, 2006). Usage control and Evidences in XACML are supported by extension as in (Colombo, et al., 2010) and (Ardagna, et al., 2009) respectively. Essentially, XACML is seen as a viable choice for PBM in pSHEILD given its concrete standardization and its support for access control policies besides other relevant features.

6.4 Affiliated protocols

A set of protocols that are usually affiliated with the management and propagation of policy interpretation to the lower level of resources are presented here.

6.4.1 COPS

Common Open Policy Service (COPS) protocol governs a client/server form of communication for policy control on QoS signalling protocols, especially between a Policy Decision Point (PDP)/server and a Policy Enforcement Point (PEP)/client. COPS is defined by IETF's RFC 2748 (IETF, 2000) and it supports two modes of operation; COPS Provisional Model (COPS-PR) and COPS Outsourcing Model. The latter is COPS' simple form of operation where a PEP sends requests for policy decisions to a dedicated PDP. The PDP then evaluates the request against a given policy and responds to the sending PEP with the applicable policy decision. In the COPS-PR mode, PEPs will inform the PDP about their ability to take policy decisions including local specifications. The PDP hence uploads the provisioned policies onto the PEP which stores them in a local Policy Information Point/Base (PIP/B) and carries out requests for policy decisions locally. While online, the PDP can update or remove policies on the PEP if it found that necessary due to some external changes. Also, PEP is expected to send a request for update to the designated PDP if its local configurations have changed where the PDP will upload any additional provisional policies on the PEP.
COPS uses TCP in order to exchange messages between clients and servers. It can also use existing security protocols such as IPSEC and TLS to support an authenticated and secure channel between PEP and PDP. Moreover, COPS is stateful as a request/decision state is shared between the PDP and the PEP as well as that pairs of requests and decisions (also referred to as events) can be interrelated.

6.4.2 SNMP

Simple Network Management Protocol (SNMP) (IETF, 1990) is a well-established protocol (approved in 1990) for monitoring entities or devices in a TCP/IP network. SNMP allows for different network management tasks such as auditing, performance monitoring, faults detection and remote configuration. It is designed to be simple where its deployment on several managed devices will not be resource demanding while still being robust. Latest versions of SNMP improved security and data integrity by using MD5 hashing and DES encryption in addition to protection against reply attacks through authentication.
As an alternative to COPS, SNMP can be used for governing the communication channel between the PDP and PEP. For instance, SNMP was recommended in the early stages of PBM to be used in network management as in (Boros, 2000). Also, in (Rana, et al., 2009) SNMP and COPS were recommended in the management of home area networks through PBM.

6.4.3 LDAP

Lightweight Directory Access Protocol (LDAP) (IETF, 2006) is an application protocol that allows for accessing distributed directory services that follow the X.500 model. LDAP is a variation of X.500 customized to provide a lightweight implementation with TCP/IP support. It is considered to be cross-platform suitable for read-intensive operations hence allowing for directory access from different platforms and locations with infrequent update requirements. From a PBM perspective, the repository used to manage stored policies can serve as an LDAP server where PDP and PAP are seen as LDAP clients. In different research concerning PBM in networks management on different aspects, LDAP is recommended for accessing policy repository (Verma, 2002)(Sloman, et al., 2002)(Flegkas, et al., 2002). An open source implementation, namely, OpenLDAP is available for free at (OpenLDAP, 2011).

6.5 Reflection on pSHIELD

In this section, a typical PBM architecture is mapped to pSHIELD's general architecture. The latter includes two types of nodes at the button node layer that are categorized based on their capabilities in terms of processing power and capacity, i.e., power nodes and sensor nodes. Power nodes are described to be more resourceful while sensor nodes are typically seen as resource constrained devices. Upper supporting layers constitute network, middleware and application layers while agents in a vertical overlay monitor/tune those layers. Given the aforementioned architecture, a PDPs and PEPs from a typical PBM architecture can be mapped naturally to power and sensor nodes respectively. Figure 37 presents the proposed PBM mapping.
On the lower layer, sensor nodes being the managed resources are considered as policy enforcers, i.e., PEPs. The latter, based on the XACML model, should enforce authorisation decisions and handle affiliated obligations specified by applicable rules. PEPs can support local policy storage in order to comply with COPS-PR mode of operation hence the provision of a local PIP although not compulsory. However, this depends on the capabilities of deployed sensor nodes whether they can afford a form of local policy storage and decision making. Moreover, power nodes are those nodes that are more resourceful than the sensor nodes which make them natural decision making points able to process/translate policies and deduce rules to be enforced by affiliated PEPs. The COPS protocol can govern the communication between PDPs and affiliated PEPs but not exclusively as SNMP is an option as well (where an LPIP is no more required).

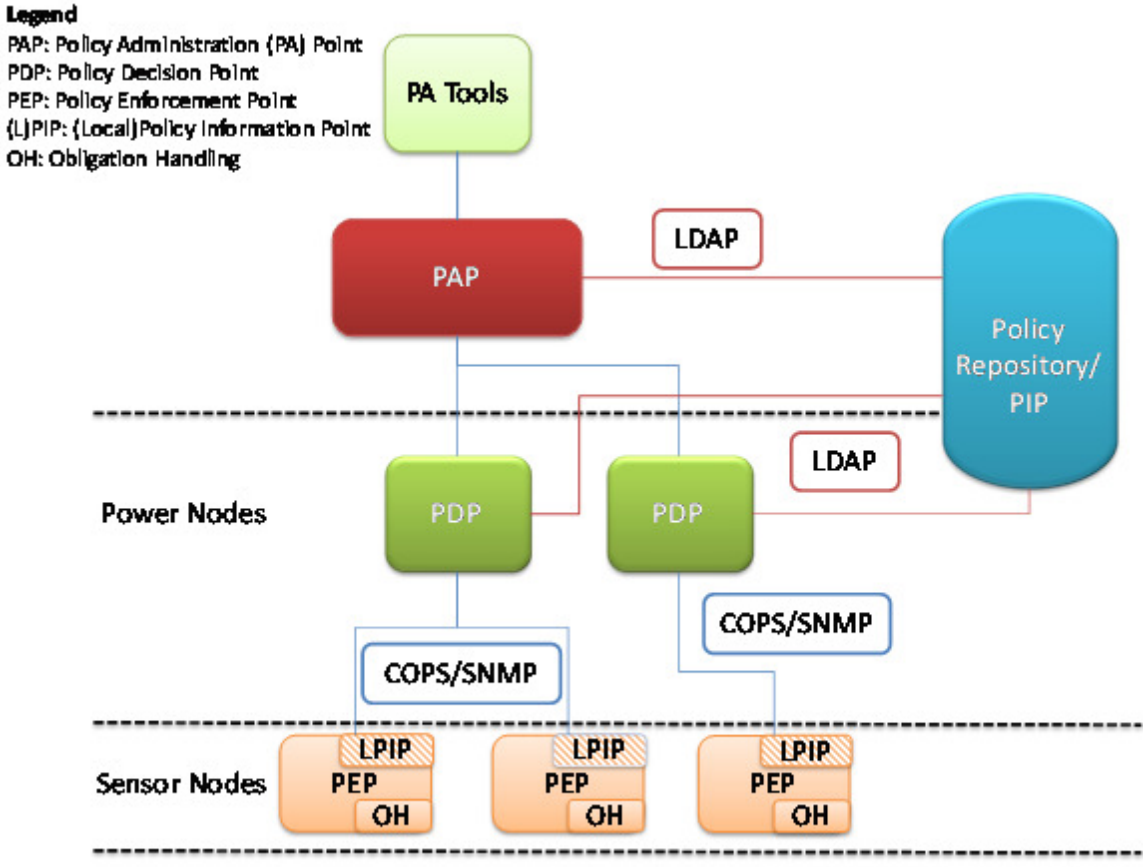

Figure 37 PBM Mapping

A group of PDPs can access the repository of policies, (i.e., PIP) in order to retrieve needed polices for evaluation. This is done through LDAP that is a protocol suited for lightweight read-intensive operations allowing for directory access from different platforms and locations. The policy repository is managed solely by the policy administrator point (PAP). Also, PAP is responsible for providing policy authoring tools besides management and control capabilities. These could include creation, termination, activation, listing, amending and synchronizing policies. Commercially for instance, Cisco's PAP (Cisco) manages,

administer and monitor policies in a central manner compatible with LDAP. It also provides several features such as web-based editing and composing of policies in a drag and drop manner, policy scope definition, delegation management and aiding functionalities for understanding the implications of policy changes. Concerning, LDAP implementations, an open source version is available as mentioned earlier at (OpenLDAP, 2011). However, COPS does not seem to have an open source implementation where some commercial ones could be available. If SNMP is considered as an alternative for COPS, some open source and free implementations are available such as in (NET11). XACML is indeed the policy specification language to be used as argued in earlier discussion.
Concerning pSHIELD's main scenario where a monitoring and access control system is put in place to oversee rail-transported hazardous materials, the above PBM is considered suitable. Locking and access control mechanism in addition to installed sensors can be seen as PEPs where the central control unit in the train carriage can be seen as a PDP with local access to PIP. Moreover, the central command centre overseeing the operation of the monitoring system is seen as a PAP with policy administration tools and repository support. The PIP is expected to be distributed which allows a given PDP to access it locally where a PAP can manage such a distributed PIP through LDAP.

6.5.1 Performance evaluation

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

The processing time depends on:

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

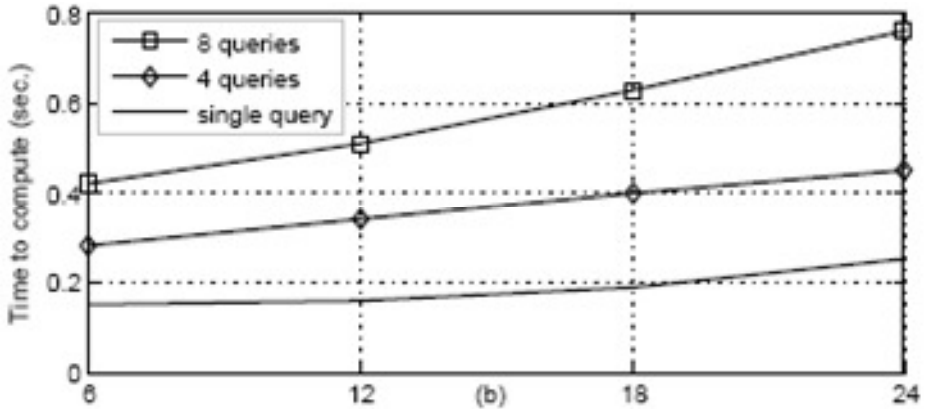

Figure 38 N° of instances/class in Knowledge Base

Figure 38 shows the computation time for simultaneous policy execution with the following simulation parameters:

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

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

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

6.6 Conclusions

This section presented and discussed technologies related to Policy-Based Management (PBM). This included policy specification languages and models in addition to affiliated protocols. Based on the discussion of available PBM technologies, a suggested PBM mapping was presented on pSHIELD's general architecture. XACML is recommended for policy specification given its standardized nature with access control support. Moreover, LDAP is found to be widely used for governing communication between PDP and PAP from one side and the repository of polices on the other side. Also, COPS or SNMP are both widely used as protocols for communication between PEPs and PDPs.

7 pSHIELD Overlay and control algorithms: the security agent embedded intelligence

7.1 Introduction

With reference to the pSHIELD functional view (see deliverable M0.1) the following figure highlights the key functionalities and interfaces involving the Overlay layer.

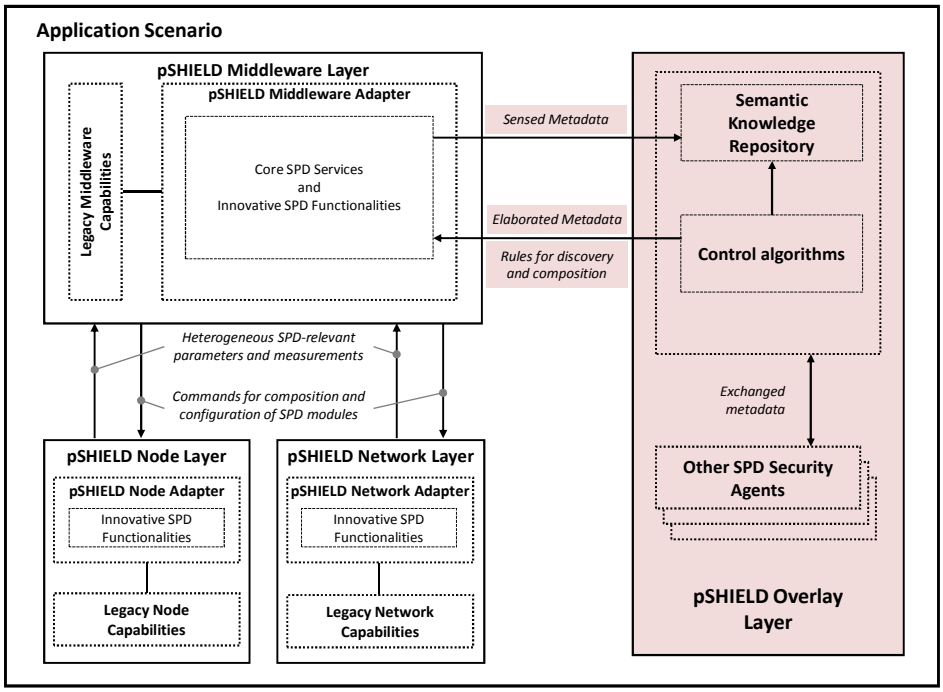

Figure 39 pSHIELD overlay: a functional view

The Overlay consists of a set of SPD Security Agents, each one controlling a given pSHIELD subsystem. Subsystem identification has to be carefully performed scenario by scenario. Expandability of such framework is obtained by enabling communication between SPD Security Agents controlling different sub-systems. As a matter of fact, the presence of more than one SPD Security Agent is justified by the need of solving scalability issues in the scope of system-of-systems: exponential growth of complexity can be overcome only by adopting the policy of *divide et impera* (divide and rules).

Each SPD Security Agent, in order to perform its work, exchange carefully selected information with the other SPD Security Agents, as well as with the three horizontal layers (node, network and middleware) of the controlled pSHIELD subsystem. However, in the scope of the pilot project the interaction between different security agent is not taken into account, since the controlled subsystem is reduced and only the "single security agent" case is considered.

Each SPD Security Agent collects properly selected heterogeneous SPD-relevant and context measurements and parameters coming from node, network and middleware layers of the controlled pSHIELD subsystem; this information is used as valuable input for the Control Algorithms (see next paragraph). Since the SPD Security Agent is mainly a software functionality and not a physical system itself, it needs the mediation of the pSHIELD Middleware Core SPD Services as it has been explained in the previous section.

Summarizing, each SPD Security Agent consists of two key elements:

(i) the Semantic Knowledge Repository (i.e. a database) storing the dynamic, semantic, enriched, ontological aggregated representation of the SPD functionalities of the pSHIELD subsystem controlled by the SPD Security Agent;

(ii) the Control Algorithms generating, on the grounds of the above representation, key SPD-relevant decisions (consisting, as far as the Composability feature is concerned, in a set of discovery, configuration and composition rules).

In deliverable D5.1 as well as in the SPD Core Services section of the current document, the first point has been widely exposed by means of prototypes. The purpose of this section is to formalize the behaviour of the Overlay and the SPD Security Agent with respect to the control algorithms (often referred to as “embedded intelligence”).

7.2 Overlay Behaviour

The behaviour of the Overlay is provided in Figure 40: thanks to the pSHIELD metrics identified in WP2, the desired SPD level can be quantified and translated into a reference signal for a closed loop control scheme. Two potential paths can be followed to achieve this desired level: (i) the first one is by means of a **policy based management**, with a set of pre-determined actions and rules (policies); (ii) the second one is by means of the **common criteria approach** enriched with **Hybrid Automata control algorithms**.

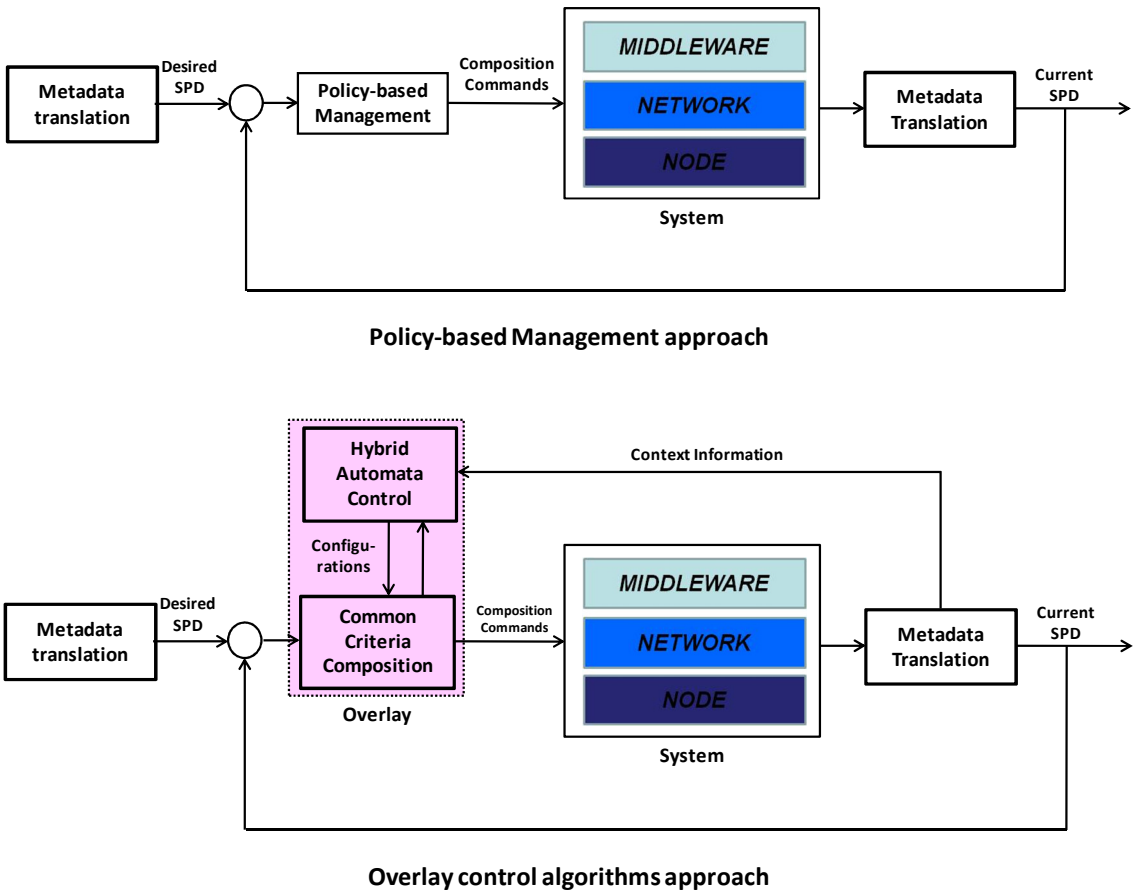

Figure 40 Overlay behaviour

The first way (the Overlay acts as policy based manager) has been analyzed in the previous section with the performances consideration already carried out.

The second way will be shown to be more efficient and to provide the best performance, but at the same time requires a biggest effort to be developed in the real system. For that reason the implementation of this solution, in the pilot project, will be limited to a simple (static) case and a set of simulations, while the theory and the rationale behind it will be addressed in this section. This procedure can be summarized in the following steps:

1) The *semantic representation* of the system (including the functional dependencies between the components as well as the atomic values of SPD for each element) *is discovered* (measured) to provide the metadata necessary to the Overlay.

2) This knowledge, thanks to the *inferential engine* described in D5.1-3 and in D2.2.1 as Common Criteria approach, is used to find a list of *configurations* of pSHIELD components that *satisfies the user requirements* in terms of SPD. Since the ontology models the functional dependencies of atomic components, all the *identified configurations are feasible*.

3) Then the semantic knowledge is used to create a model *of the system and its evolution* by means of *Hybrid Automata Theory* (see next paragraph) and this model, enriched with *optimization* control algorithms, is used to decide which, among the suitable configuration, is the *best*, with respect to context information.

These logical flow is depicted in Figure 41.

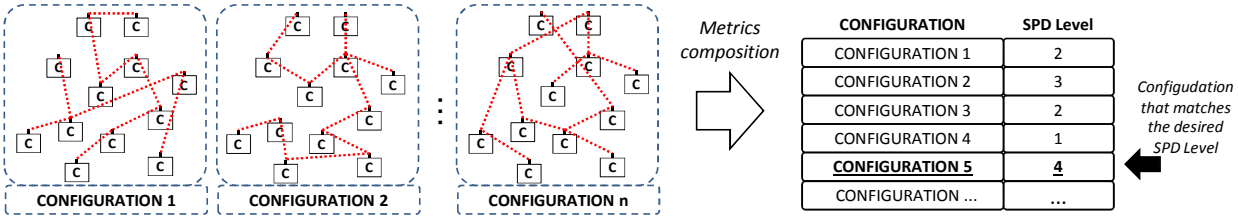

Figure 41 Overlay approach for configuration identification

The semantic representation (step 1) has been widely addressed in deliverable D5.1-3 and in the previous section.

The rules and mechanism to compose metrics to measure the configuration SPD level (step 2) are available in deliverable D2.2.2 as well as in the reasoning section of D5.1-3.

The configuration choice basing on context information is the objective of the following paragraph.

7.3 The "context aware" mechanism to drive the SPD composability in ESs

Modelling the **composability** is a challenging issue faced by different research activities both in academic and in European Project environments. This is the case, for example, of the European project *Connect* (http://connect-forever.eu) or *EternalS* (http://www.eternals.eu) whose aim is to develop an abstraction framework for the "eternal" connection of evolving networked devices and/or software functionalities.

In these context the most common methodologies to model the composition of elements (on a theoretical point of view) are widely used: Petri-Nets, Bayesian Networks, Graphs, Markov Chains, and so on.

These methodologies have been the bases of the technological scouting performed by pSHIELD in this field as well, to check the feasibility of these solution. However all the classical mathematical and logical methodologies have shown not to be adequate to model the pSHIELD contex due to two peculiarities:

1) pSHIELD is an **heterogeneous** system, so it is necessary to find a theory able to model "heterogeneous elements" in a single entity at a time.

2) pSHIELD is a **dynamically composable** system, so the modelling tool should be scalable and composable as well (no unique information/energy flow among components)

Furthermore, this model should be ready to be integrated and implemented in analog/digital closed loop control schemes, meaning that the resulting composition should **model the evolution** of the system (in nominal condition or in presence of faults).

For all these reasons, the adopted approach will be based on Hybrid Automata: a mathematical framework by which it is possible to model Hybrid Systems (continuous/discrete/heterogeneous domains like Embedded Systems) compose them (parallel composition) and make them evolve (automaton).

Different models have been tailored for the pSHIELD purposes and in the following paragraphs two of them will be presented: one static, simpler but less scalable, and one dynamic, more complex but scalable and more expressive.

Before continuing, a short recall of what is an Hybrid Automata is provided (see [1][2])

7.3.1 Hybrid Automata

An *Hybrid Automaton (HA)* $A = (W, X, Q, \Theta, E, H, D, T)$ is an entity composed by:

- A set of external variables *W* as well as internal variables *X*, disjoint $V \triangleq W \cup X$
- A set of finite states $Q \subseteq val(X)$.
- A set, non empty, of initial states $\Theta \subseteq Q$
- A set *E* of external actions and a set *H* of internal actions disjoint $A \triangleq E \cup H$
- A set of discrete transitions *D* and a set of trajectories *T*

To describe the behaviour (evolution) of an automaton, it is sufficient to indicate the starting state, the transitions (with or without resets) and the next state. Depending of the time horizon, the starting states and the external actions, this description is done by means of: *execution fragments, executions, trace fragments* e *traces*.

The most powerfull feature of Hybrid Automata is that they can evolve independently, but can be composed together to exchange Input/output and to drive the evolution of each other transitions. This is known as *parallel composition*.

In the scope of the pSHIELD project, some Hybrid Automata have been designed to model the pSHIELD generic component, composed together and controlled.

7.3.2 Soultion A – Static Approach with Simple Optimization

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

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

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

The state identified in this step is depicted in Figure 42.

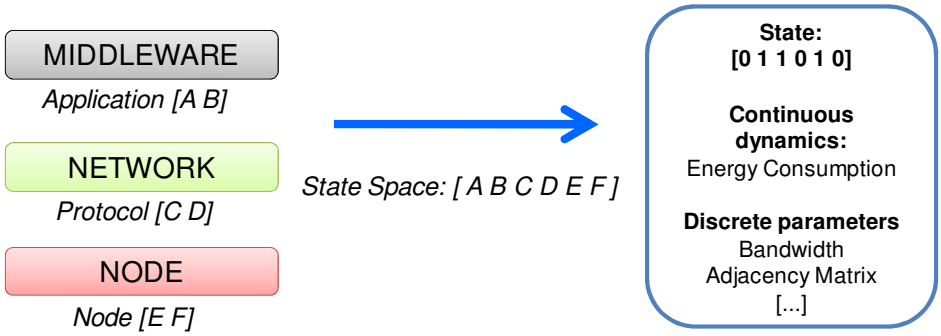

Figure 42 Single State

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

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

The result is depicted in Figure 43

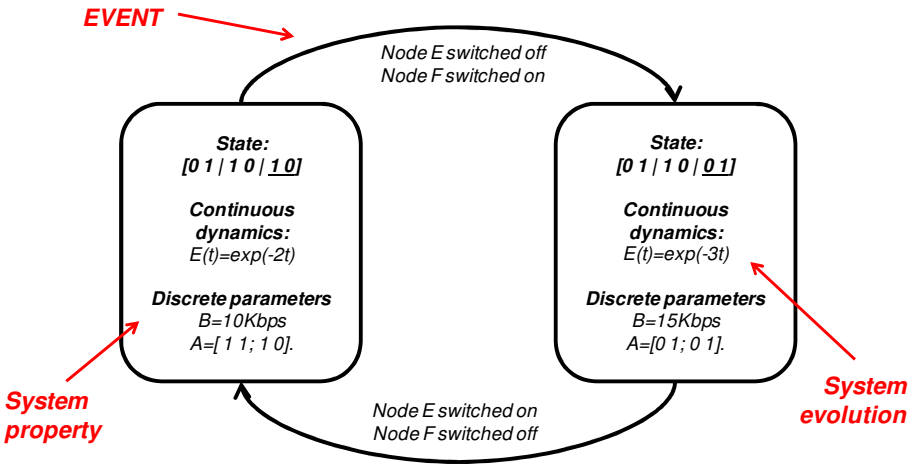

Figure 43 Hybrid Automata to describe all the possible configurations

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

D = {switch configuration$_1$, fault$_1$, …, switch configuration$_n$, fault$_n$}.

The third step is the indentification of the internal variables (and dynamics) to control. For the pilot project a simple case is considered where the relevant dynamic is the power consumption of the system in a specific configuration and the amount of bandwidth provided by the network layer. These variable have opposite behaviours (higher bandwidth, higher power consumption) so the purpose of the control algorithm is to choose the configuration that optimizes one of them.

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

A simple controller is in charge to switch to the configuration that guarantees the longest duration of the node batteries

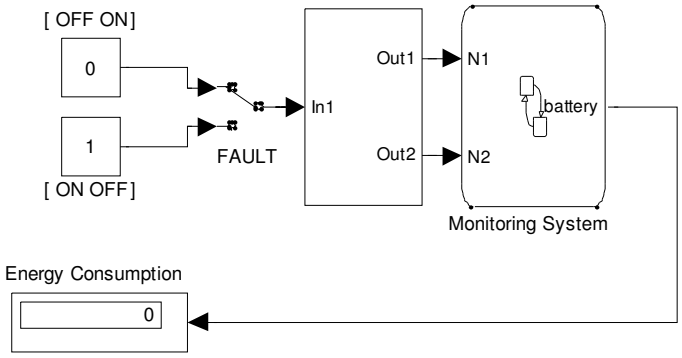

Figure 44 Simulink model

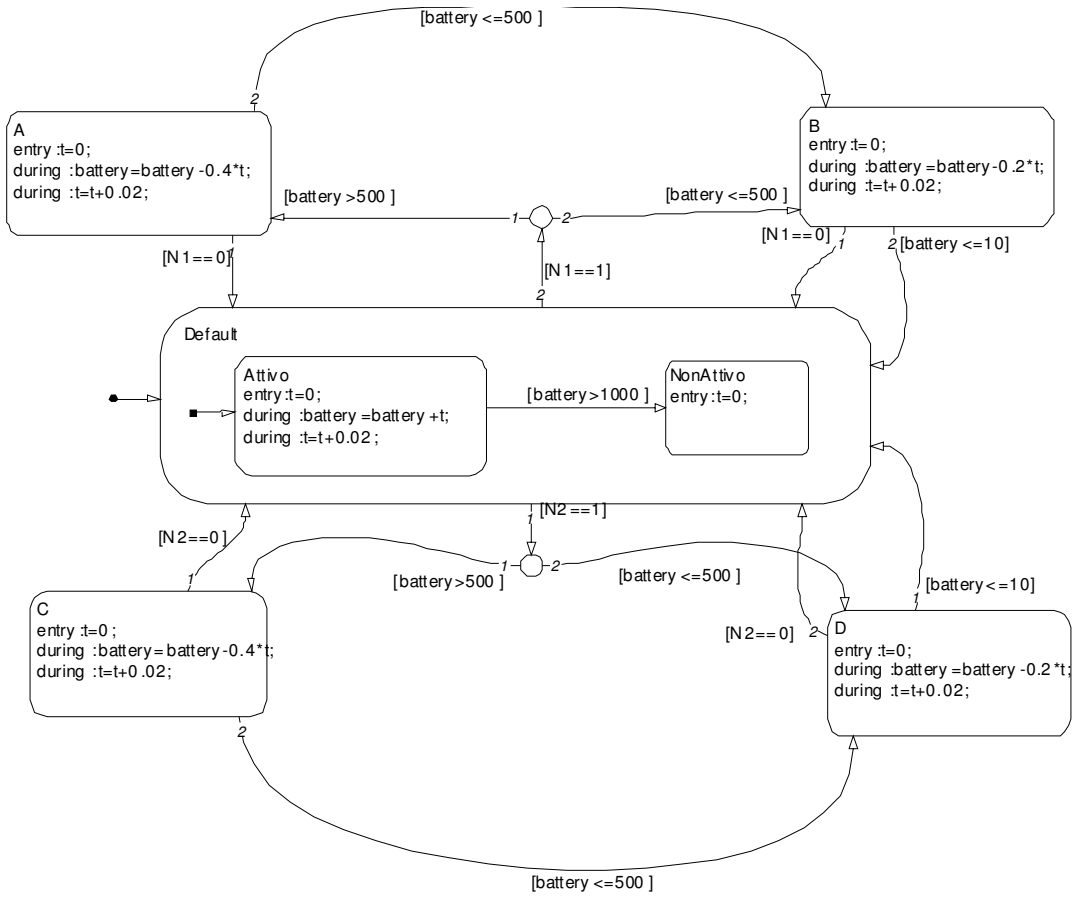

Figure 45 Hybrid automata with four states

In Figure 46 the result of the simulation is carried out: when the power consumption is not a problem, the system remains in the configuration 'A' that allows him to use all its resources: when the dotted line (energy consumption reference) is lowered, the node switch in a saving configuration 'B' and starts wasting less power, thus lasting more.

Since many parameters can be included in the continuous dynamic, a multi-objective optimization can be performed to obtain optimal performance driven by context information.

On a deployment point of view, this model could be implemented in an intelligent node that can simulate the system behaviour and adequately react to faults and evolutions.

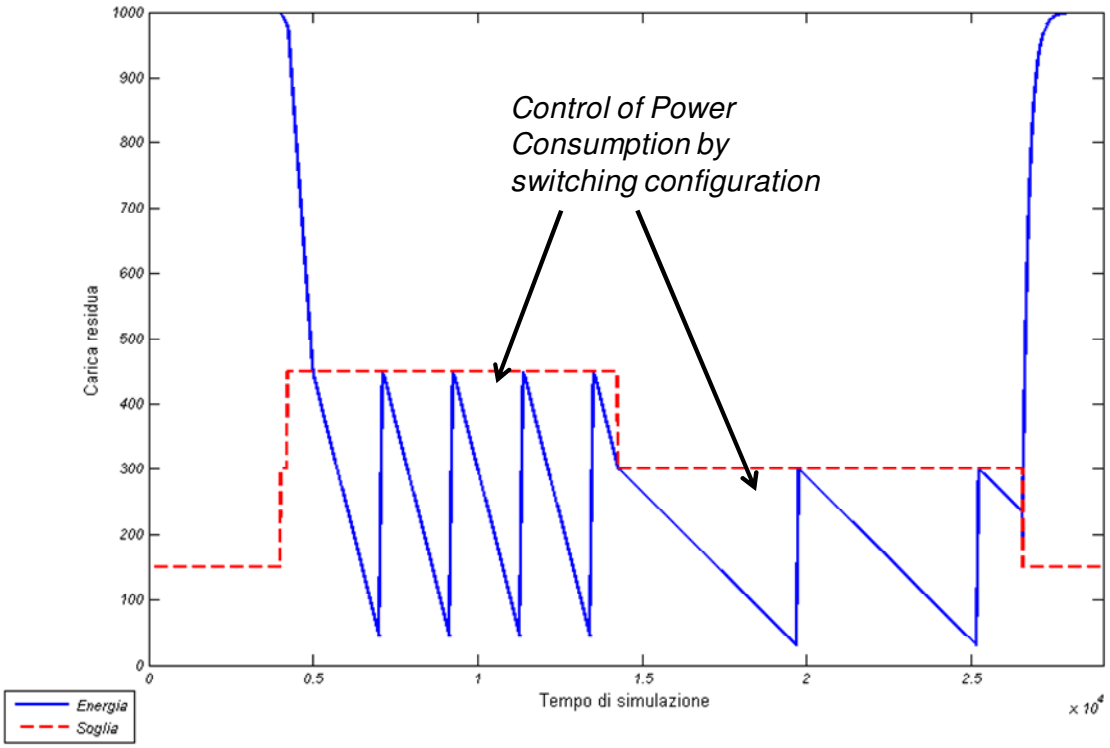

Figure 46 Configuration switching to optimize piower consumption

Due to its simplicity, this approach suffer for a scalability problem, since when the number of component rises, the number of possible configurations grows exponentially. A more intelligent and efficient approach is needed to solve the problem of scalability.

7.3.3 Solution B – Operating conditions approach with MPC Control

A better solution has been found starting from the analysis, in literature, of the work carried out by Balduzzi (see [4][5]) to model manufacturing environment by means of Hybrid Automata. His idea is very simple: in a complex production system ,where all the machines (elementary services) are connected to produce goods, only the operating conditions of the machines influences the system behavior. Moreover the dynamic of the machine changes only when the operating conditions change. For example a machine could be broken, operative in linear conditions or operative and saturated. By using these conditions as "states" of the hybrid automata, it is possible to describe in a simple but expressive way, all the relevant dynamic of the system and the control it. A sample screenshot taken from [4] is depicted in Figure 47.

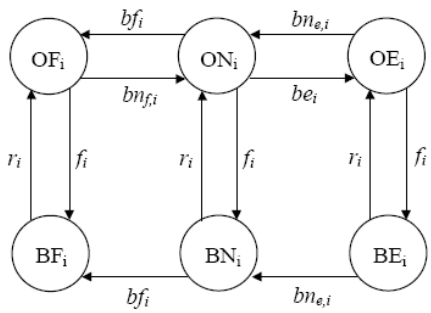

Figure 47 Operating condition of a manufacturing system as modelled in [4]

Given an Embedded System (pSHIELD Node) it is possible to identify a set con elements (battery, buffers, CPU) that can be associated to an operating conditions: a buffer can be saturated, full or empty; a CPU can be idle, working or overloaded; a battery can be full or empty. All these components can also be broken. The combination and aggregation of these conditions allows to create an exhaustive model of a pSHIELD node, as depicted in Figure 48. The aggregation is possible, since some behaviours of the components have the same effect of the system (if the CPU or the Buffer is full, the result is always the impossibility of processing data).

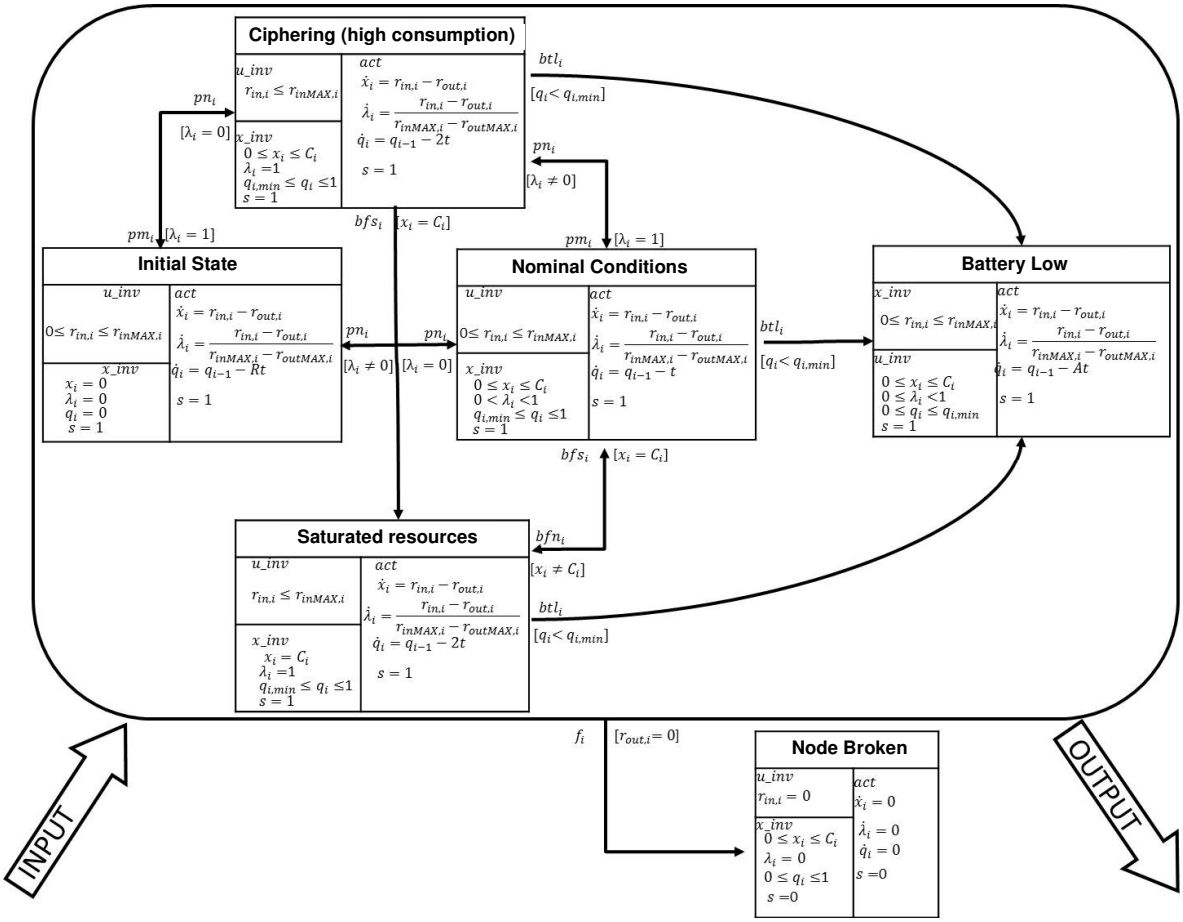

Figure 48 Hybrid Automata representing the pSHIELD node

In each state, a simple, but effective, set of parameters and dynamics has been identified to describe the generic behavior of a pSHIELD node:

Buffer: the node usually receives in input a set of data, hence it contains a buffer that can be empty, partially full or saturated (and for each case a specific operating condition occurs: initial state for empty buffer, nominal condition for partially full buffer and saturated resources for completely full buffer).

The dynamic that indicates the evolution of the buffer level is the difference between incoming and outcoing rate of data to/from the node (being it a sensor, an actuator or a relay node).

$$x_i(t) = x_i(t_k) + [r_{in,i}(k) - r_{out,i}(k)](t - t_k) \quad (1)$$

CPU: the node elaborates information by means of its computational capabilities (usually one or more CPUs). In a similar way, the CPU can work normally or can be overloaded. The amount of elaboration capability being used depends on the amoun of data elaborated by the node (in/out rate). This value is a percentage on the maximum allowed rates. Moreover, depending on the type of elaboration, a scaling factor may be included (i.e. strong information ciphering results in higher used resources).

$$\lambda_i(t) = 100\,\frac{r_{in,i}(k) + r_{out,i}(k)}{r_{inMAX,i} + r_{outMAX,i}}(t - t_k) \quad (2)$$

Battery: the node is equipped with a power supply (usually a battery, but a wired power source may be considered as an infinite power battery). Depending on the amount of incoming/outcoming data as well as the amount of elaboration, the power consumption is quicker or lower. The consumption dynamic is very simple, is linear, and depends on the load R.

$$q_i(t) = q_0 - R(t - t_k) \quad (3)$$

Both CPU and Battery can be in different working conditions: saturated, nominal, broken… each one captured (or coupled) in the above depicted Hybrid Automata.

Transitions and constraints are simple too and can be set by reading the ontology describing the node (maximum buffer size, maximum battery charge, maximum CPU power, and so on).

The same process can be done for the Network Layer, since the only relevant operating conditions for a network (at this stage) are two: its *congestion status* and its *connectivity* (i.e. the possibility of reaching all the nodes). A four state model is obtained (congestioned and connected, congestioned and not connected, connected and not congestioned, not connected and not congestioned). See Figure 49 for the model.

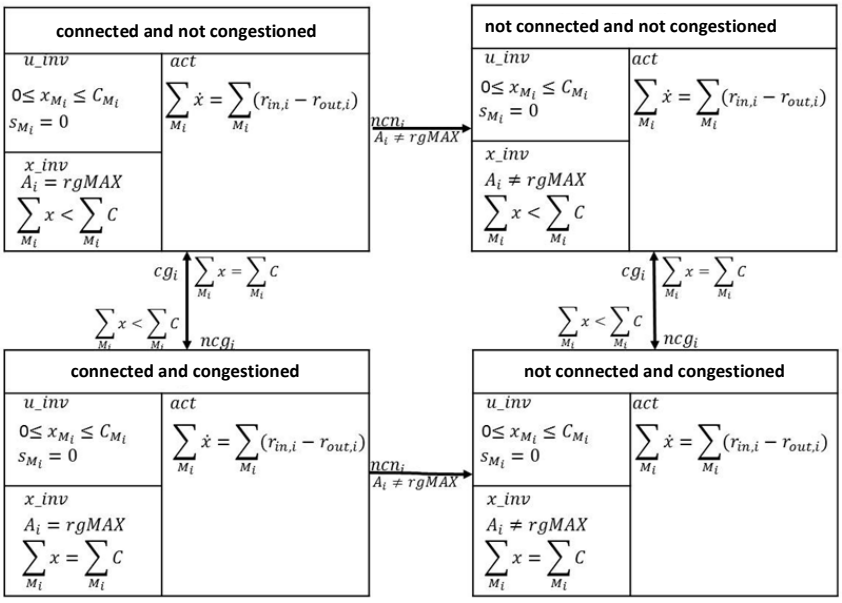

Figure 49 Hybrid Automata representing the pSHIELD network

The dynamics are heritage of the nodes and merely the sum of the in/out rates involved in the network.

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

Last, but not least, interesting control algorithms can be applied to the system model due to its formulation by means of these operating conditions (see for example the work of Bemporad [6] and [7]). In particular for the pSHIELD purposes the framework developed in [7], based on Model Predictive Control (MPC), has been considered to verify the effectiveness of the Hybrid Automata approach.

7.3.3.1 MPC Controller

One of the most suitable control law to implement on this Hybrid Automata framework is the Model Predictive Control (MPC). This control methodology is based on the computation, step by step, of the optimal input to the system to achieve a certain objective (usually the minimization of a cost function under certain constraints). This input is a train of control signal from the starting sample time to infinity and is updated each subsequent sampling time.

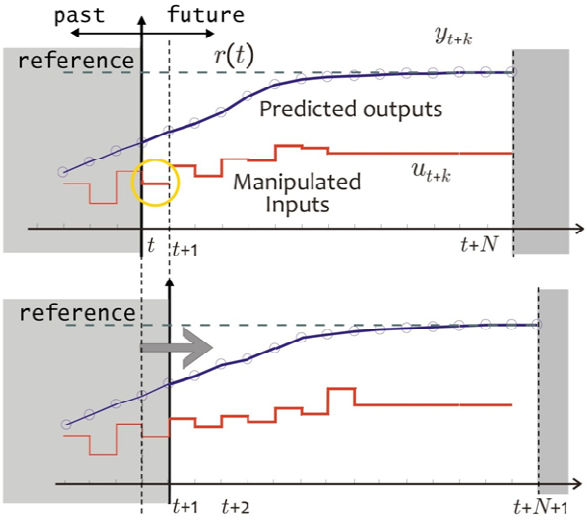

Figure 50 Model Predictive Control

This approach is suitable for many reasons:

- it perfectly fits sampled systems
- in case the control action cannot be computer during a step, the previously obtained control law is still valid until update
- it is suitable to control MIMO systems with multiobjective optimization

Due to many similarities in the problem formulation, for the pSHIELD proof of concept the same approach of [7] has been adopted. In the following the result is reported, by using the pSHIELD Hybrid Automata matrixes in the formulation.

The MPC problem is formulated with a quadratic (or H inifinity) cost function that aims at minimizing the error and/or the control power and/or the state and/or the output (depending on the weigth on the cost function components):

$$\min_{\{v,\delta,z\}_0^{N-1}} J(\{v,\delta,z\}_0^{N-1}, x(t))$$
$$\triangleq \|Q_{xN}(x(N|t) - x_r)\|_p$$
$$+ \sum_{k=1}^{N-1} \|Q_x(x(k) - x_r)\|_p$$
$$+ \sum_{k=0}^{N-1} (\|Q_u(v(k) - u_r)\|_p + \|Q_z(z(k|t) - z_r)\|_p + \|Q_y(y(k|t) - y_R)\|_p$$

Under the constraints defined by the system model

$$\begin{Bmatrix} x(0|t) = x(t) \\ x(k+1|t) = Ax(k|t) + B_1 v(k) + B_2 \delta(k|t) + B_3 z(k|t) \\ y(k|t) = Cx(k|t) \\ E_2 \delta(k|t) + E_3 z(k|t) \leq E_1 v(k) + E_4 x(k|t) + E_5 \\ u_{min} \leq v(t+k) \leq u_{max}, \quad k = 0,1,\dots,N-1 \\ x_{min} \leq x(t+k|t) \leq x_{max}, \quad k = 1,\dots,N \\ y_{min} \leq y(t+k) \leq y_{max}, \quad k = 0,1,\dots,N-1 \\ S_N x(N|t) \leq T_N \end{Bmatrix}$$

This formulation is simple and in line with the one defined in [7], but it works perfectly as a proof of concept.

For the simulations it has been used the Matlab Toolbox for Hybrid System with the default configuration (standard MPC problem). The Objective of the control algorithm has been to maximize the amount of data data processed by the node while preserving the battery and leaving a certain amount of "reserved" resources for potential emergency tasks. This objective is "ambitious" but the power of MPC control is the multi-objective optimization over a temporal horizon, and the results of Figure 51 demonstrate that the system follows the desired behaviour and all the objectives are met, On a practical point of view this is translated into a set of control commands from the overlay to the system, that decide time by time which component should be activated to satisfy the requirement.

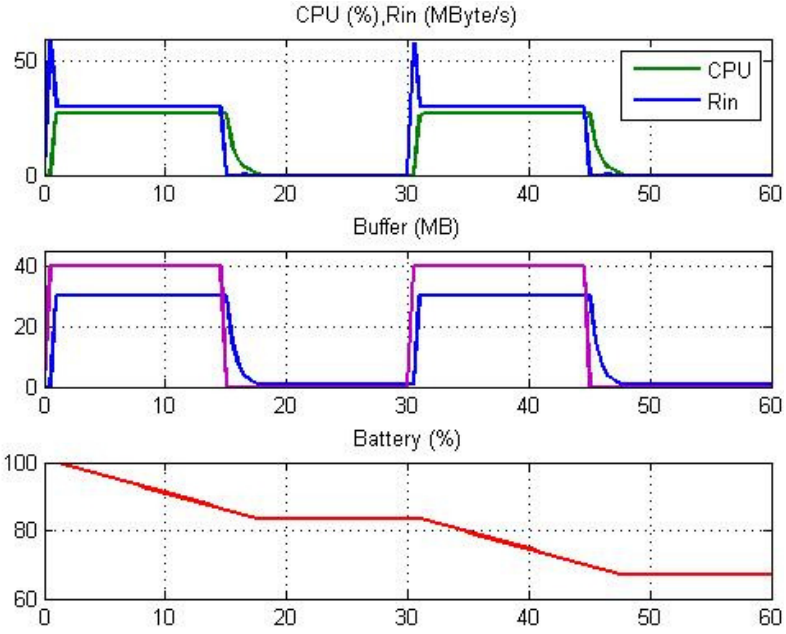

Figure 51 System behaviour with MPC control

Also in this case it is important to notice that all the behaviour and the configurations are considered to be “feasible”, so the objective of the controller is to tailor parameters or switch component, according to context information, to perform context optimization (even not directly connected to SPD issues).

In the subsequent studies (nSHIELD project) this control algorithm will be refined and embedded in different agents and proper communications mechanisms and interactions will be defined to enable a more complex behavior. In fact, the overlay is populated by a set of entities (the security agents) each one implementing control tasks.

In this perspective, in the last paragraph some interesting analysis on the role of semantics in agent based systems are reported.

7.4 Reactive Agents

RDF, RDFS, and OWL languages enable us to create Web-based knowledge in a standard manner with a common semantics. We now turn our attention to the techniques that can utilize this knowledge in an automated manner. These techniques are fundamental to the construction of the Semantic Web, as without automation we do not gain any real benefit over the current Web. There are currently two views of the Semantic Web that have implications for the kind of automation that we can hope to achieve:

1. An expert system with a distributed knowledge base
2. A society of agents that solve complex knowledge-based tasks

The programming model envisioned for the Semantic Web is based on the notion of software agents as the key consumers of knowledge. These agents act collectively as a community, rather than as a single entity. The agents are responsible for the collection, processing, and dissemination of knowledge in order to achieve specific goals. Agents exhibit a form of intelligent behaviour, in that they rationalize and can act autonomously.
A community of agents that act together is termed as MAS. This is the main concepts in the construction of MASs. We identified two key considerations that must be addressed in any such system, which we summarize below:

1. The first consideration is the design of the internal *reasoning processes* of an agent. In essence, how we construct programs that can reason about the world in similar ways to humans
2. The second consideration is the design of the *communicative processes* of the agent, as we want to define agents that can interact. To construct agents that can interact, we typically take our inspiration from the study of human dialogue

A key consideration in the definition of any agent system is the *environment* in which the agents will operate. Multiagent have been designed for a wide variety of different environments, e.g. robots, sensor networks, embedded systems. In this book, the environment of the agent system is the Semantic Web. For this reason, we restrict our attention to the definition of agent systems that can operate in this domain. In particular, the assumptions that we make in the design of our agents are directly related to the capabilities of the Web and the Internet.

7.4.1 Rational Agency

A *rational* agent is one that acts to further its own interests in light of its beliefs about the world. For example, if an agent has the goal to purchase a camera at the lowest price, and it believes that the lowest price is available at PhotoMart, then it would be rational for the agent to obtain the camera from PhotoMart. The agent would be *irrational* if it decided to purchase the camera elsewhere in light of the stated goal, and its beliefs. An agent can still be considered rational if it acts in accordance with erroneous beliefs.
This raises the important issue of how to design computational agents that can act rationally.

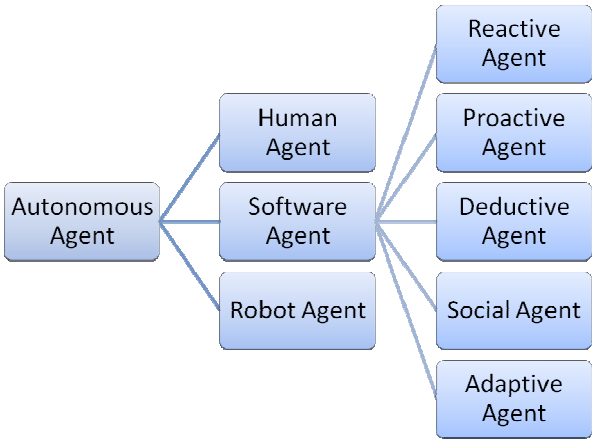

Figure 52 An ontology of agent properties

A basic ontology for describing the properties of an autonomous agent is presented in Figure 52. Our interest here is restricted to software agents, and therefore we have only expanded this part of the ontology. An agent instance would typically be ascribed more than one property, e.g. a reactive and proactive agent.

The properties that we have discussed give us a favour of the sort of programs that we are seeking to construct. As we have previously stated, there is no single definition of the term *agent*. The actual definition and implementation of the agent is specific to the kind of task that we are seeking to perform. All that we require is that our agent can interact with other agents, and do useful work for us. We cannot hope to describe all of the possible approaches to implementing agent systems.
We consider four distinct styles of agent implementation: *reactive* agents, *practical reasoning* agents, *deductive reasoning* agents, and *hybrid* agents. These styles can be considered as representing the most popular proposals for rational agency.

7.4.1.1 Distributed Agent Systems

Agent systems are necessarily distributed and concurrent in nature.

1. We have highlighted autonomy as a key property of an agent. Therefore, it is assumed that the agent makes decisions about synchronization and coordination at run-time. However, in a distributed system, these mechanisms are typically hardwired by design.

2. Rational agents are primarily self-interested entities. Therefore, it is assumed that an agent is primarily interested in furthering its own goals on behalf of a particular user. In a distributed system, the entities are generally assumed to share a common goal.

7.4.2 Reactive Agent System

The most straightforward way to construct an agent with a specific pattern of behaviour is simply to engineer all of the required behaviour directly into the agent. This is the approach taken in the construction of a *reactive agent*. A purely reactive agent does not perform any kind of deduction, and so its behaviour is easier to test and to predict, i.e. it should always act the same way in response to a given sequence of events. However, despite this apparent lack of reasoning power, the reactive approach to agent design can capture a wide range of useful agent behaviours. The agent is engineered to respond to changes in the environment, which we represent as input events. An input event causes the agent to perform some predefined behaviour that may result in an output action on the environment, e.g. a message is sent. An output action will cause a change in the environment, which may result in further input events for the agent. In effect, the agent is acting as a *sensor* on the environment. We can readily construct an MAS from reactive agents, where a single trigger event causes a cascade of events and actions between the agents.

7.4.3 Practical Reasoning and Deductive Agents

In constructing a reactive agent system, we explicitly define the behaviour of each agent. This behaviour is predefined, and dependent on events in the environment. We now consider a more powerful kind of agent that can make decisions on its own, i.e. an agent with proactive behaviour. Our motivation is the construction of agents with capabilities that are closer to the way that we reason as human beings. Our starting point in this approach is to base the internal processes of the agent directly on current understanding of how human reasoning is performed. This is the principle behind the design of a *practical reasoning agent*.
Practical human reasoning is directed towards actions, that is, figuring out what to do. This is different from purely logical reasoning, which is directed towards beliefs. Human reasoning is believed to consist of two distinct phases:

- The first phase is *deliberation*, in which we decide what state of affairs to achieve.
- The second phase is *means–ends reasoning*, in which we decide how to achieve the desired state of affairs.

To better illustrate human reasoning, it is helpful to consider a small example. Suppose that I wish to find a method of transportation in order to get to work each day. I would typically proceed by considering the various available options and weighing up the pros and cons. For example, I may consider travelling by car, but the available parking may be insufficient. This process of decision-making is deliberation. Once I have fixed upon an appropriate method of transport, e.g. by bicycle, then I must decide how to bring about a situation where this method of transport is possible. This process is means–ends reasoning, and the result is a plan of action. For example, my plan may involve: obtaining the money for the bicycle, finding a shop that sells an appropriate bicycle, and then purchasing the bicycle. If I am able to successfully follow the plan, then I will have reached the intended state of affairs, i.e. I will be able to travel to work by bicycle.

A practical reasoning agent attempts to replicate the process of human practical reasoning by performing deliberation and means–ends reasoning as computational processes. These processes can be defined using AI planning technology. However, the end result is only a poor approximation to real human reasoning as there are inherent limitations that arise when we move from our description of human reasoning to a computational model. We now discuss four key restrictions that we must address when designing a computational model of human reasoning.

1. Any computational model of a real world system is subject to inherent *resource bounds*. In particular, an agent will have only a finite amount of memory and a limited amount of computational power available. As a result of these restrictions, the agent is limited in the power and scope of the deliberation that can be performed. The agent will often be forced to stop deliberation prematurely and commit to a state of affairs. This may lead to a non-optimal course of action, which may have been alleviated if further deliberation were performed.

2. A Web-based agent exists in a highly *dynamic environment* that may be subject to rapid change. When the agent begins deliberation, it will typically operate on a snapshot of the world state at that particular point in time. However, as deliberation itself can take a significant length of time to complete, it may often be that the outcome will already be invalid due to changes in the environment. This requires us to make trade-offs in the design of our agents between the speed of response, and the optimality of the decision-making.

3. The reasoning process will often result in the possibility of *alternative actions*, all of which may appear equally compelling. At this point the agent is forced to make a selection based on appropriate heuristics. For example, the shortest plan, or the one that has the least associated cost. This may again result in non-optimal decision-making.

4. The final restriction that we consider is based on the need for an *abstract view* of the world. Thus, the decisions of the agent are necessarily restricted by the quality of the representation. For example, if our agent is reasoning with ontological knowledge about the world, then the deliberation is limited to the facts that can be deduced from the knowledge base. If the knowledge base contains invalid information, then the decisions that the agent makes may be incorrect.

7.4.4 Hybrid Agent Systems

We have now detailed the main techniques that can be used to define rational agents. It should be clear from our explanation that each of these approaches has certain inherent advantages and disadvantages. The reactive formalisms are relatively straightforward to implement, but they are only capable of reacting to events, and not initiating events of their own accord. The practical reasoning and deductive approaches allow us to define more advanced agent behaviours. However, they require us to have a consistent view of the world and suffer from computational complexity in implementation. Nonetheless, there is no real reason why these approaches should be considered in isolation. We can alleviate the disadvantages of any one approach by considering agents that utilize a combination of different techniques. An agent system that is constructed in this manner is termed a *hybrid* agent system.

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

agent properties. There are two main ways to structure the layers of a hybrid agent, illustrated in Figure 53:

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

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

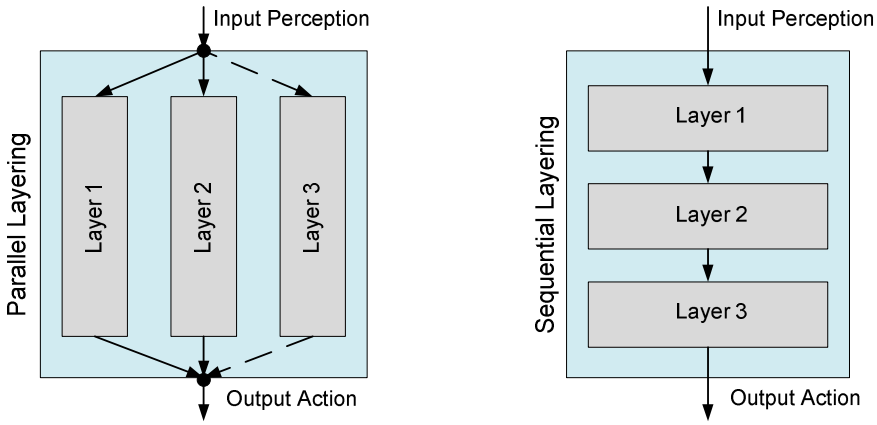

Figure 53 Hybrid Agent Architecture

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

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

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

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

At the beginning, we defined agent reasoning and agent communication as separate processes in the design of our agents. Therefore, it should now be clear that we promote the layered approach to agent design in this book. This separation makes it clear which parts of the agent are responsible for internal deliberation, and which are for external interaction. The resulting agents are arguably more dependable than an approach that combines these tasks into a single formalism. In answer to the criticisms stated

above, we argue that a unified semantics is unnecessary if we can reason about the individual layers of the system, and we define how the layers interact.

In Figure 54 we present a hypothetical hybrid agent for the Semantic Web, which makes use of all the different techniques. The agent is defined using a two-pass sequentially layered approach. At the lowest level, we have a *knowledge service* layer which is used to combine together different agents into knowledge services for performing specific tasks. Above this layer, we have a *coordination* layer which controls the coordination between the different agents cooperating on a task.

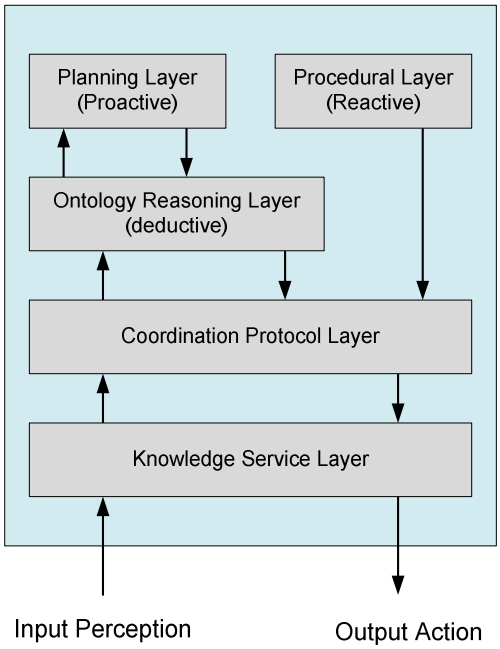

Figure 54 A semantic Web agent

The actual reasoning of the agent is defined in the higher layers. The *procedural* layer defines the reactive behaviours of the agent in terms of decision procedures. A decision procedure is a deterministic process that returns a choice on some specific input. The reasoning about the knowledge on the Semantic Web is performed deductively in the *ontology reasoning* layer. In this layer, we perform inference about the knowledge that we have available, and attempt to solve the task at hand. This layer is closely connected to the *planning* layer, which provides the proactive agent behaviour. For example, if we do not have enough knowledge available during inference, then we will construct a plan on how this information may be obtained.

We note that the term *hybrid agent* is often used to mean an agent system composed of different kinds of agents, e.g. a system that is defined with both software agents and human agents. This definition is more general than the one that we have presented here, as we are primarily interested in software agents. Nonetheless, while we do not propose to model humans explicitly as agents, it is necessary to consider how humans will interact with the agents that we define.

7.5 Conclusions

In this section an innovative formulation of control problems in Embedded Systems environment have been presented. In literature there is no model for a generic ES as well as a composition of ESs, so this analysis constitute the first step towards the formalization of a theory useful to control composability of heterogeneous devices.

Two approached have been presented: the first one is simple but not scalable, while the second one requires a biggest effort (mainly in the implementation) but assures a major scalability.

Both the approaches are used to model contex information of the pSHIELD system and to “simulate” its evolution over the time, even in response to fault or unexpected events. The objective of this formulation is to apply a control algorithm to control the value of the above mentioned context parameters, thus optimizing the choice of the system configuration.

On a deployment point of view (that will be addressed in the nSHIELD project) these models could be implemented in real time emulators running into the most powerfull node and could provide support to the overlay decisions.

8 References

[1] Henzinger T. A., "The Theory of Hybrid Automata", *Proceedings of 11th Annual IEEE Symposium on Logic in Computer Science (LICS96)*, pp. 278-292, New Brunswick, New Jersey, 27-30 July, 1996

[2] Lunze J. and Lamnabhi-Lagarrigue F., "Handbook of Hybrid Systems Control – Theory, Tools, Applications", Cambridge University Press, 2009

[3] Yang H., Jiang B. and Cocquempot V., "Fault Tolerant Control Design for Hybrid Systems", Lecturer Notes in Control and Inofrmation Sciences, Springer, 2010

[4] Balduzzi F. and Kumar R., "Hybrid Automata Model of Manufacturing Systems and its Optimal Control Subject to Logical Constraints", International Journal of Hybrid Systems, pp 61-80, volume 3, number 1, 2003.

[5] Balduzzi F., Giua A. and Seatzu C., "Modelling Automated Manufacturing Systems with Hybrid Automata", Proceedings of Workshop on Formal Methods and Manufacturing (WFMM99), Zaragoza, Spain, pp. 33-48, 6 September, 1999.

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

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

[8] Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2003). Web services. Springer.

[9] Berners-Lee, T., Hendler J., &. Lassila, O. (2001). The semantic web. Scientific American, 284(5), 34–43.

[10] Dumitru R., Uwe K., Holger L., Jos de B., Rub´en Lara, Michael S., Axel P., Cristina F., Cristoph B. and Dieter F. (2005) Web Service Modeling Ontology, Applied Ontology, 77–106.

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

[12] Pedrinaci, C., Domingue, J. and Sheth, A. (2011) **Semantic Web Services**, in eds. John Domingue, Dieter Fensel and James Hendler, Handbook of Semantic Web Technologies, 2, Springer

[13] Schulte, Stefan **:** *Web Service Discovery Based on Semantic Information - Query Formulation and Adaptive Matchmaking.* TU Darmstadt [Ph.D. Thesis] , (2010) http://tuprints.ulb.tu-darmstadt.de/2293/

[14] George Baryannis: The Frame Problem in Web Service Specifcations, University of Crete Computer Science Department [Master's Thesis] (2009). http://ics.forth.gr/_publications/Baryannis_master.pdf

[15] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara. OWL-S: Semantic Markup for Web Services. W3C Member Submission. (2004) http://www.w3.org/Submission/OWL-S

[16] P. F. Patel-Schneider, P. Hayes, I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation. (2004)

[17] Chris Metcalf C, and Grace LewisModel Problems in Technologies for Interoperability: OWL Web Ontology Language for Services (OWL-S), Technical Note CMU/SEI-2006-TN-018, April 2006.

[18] Klusch, M. (2008): Semantic Web Service Description. In: M. Schumacher, H. Helin, H. Schuldt (Eds.) CASCOM - Intelligent Service Coordination in the Semantic Web. Chapter 3. Birkhäuser Verlag, Springer. http://www-ags.dfki.uni-sb.de/~klusch/i2s/cascom_book_ch3-4.pdf

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

[20] Martin, David, et al. Bringing Semantics to Web Services: The OWL-S Approach.b(2004).

[21] World Wide Web Consortium. OWL Web Ontology Language Guide: W3C Recommendation (2004). February 10, 2004. http://www.w3.org/TR/owl-guide/

[22] World Wide Web Consortium. OWL Web Ontology Language Reference: W3C Recommendation February 10, 2004. http://www.w3.org/TR/owl-ref/

[23] World Wide Web Consortium. OWL Web Ontology Language for Services (OWL-S). (2004).http://www.w3.org/Submission/2004/07/

[24] World Wide Web Consortium. Web Service Modeling Ontology (WSMO) Submission. (2005). http://www.w3.org/Submission/2005/06/

[25] World Wide Web Consortium. WSDL-S Submission Request to W3C. / (2005) http://www.w3.org/Submission/2005/10

[26] World Wide Web Consortium. Semantic Web Services Framework (SWSF). (2005) http://www.w3.org/Submission/2005/07/

[27] CASCOM Project Deliverable D3.2: Conceptual Architecture Design. September 2005. www.ist-cascom.org

[28] D. Fensel, H. Lausen, A. Polleres, J. De Bruijn, M. Stollberg, D. Roman, J. Domingue. Enabling Semantic Web Services: the Web Service Modeling Ontology Springer (2007).

[29] D. Roman, H. Lausen, U. Keller. Web Service Modeling Ontology (WSMO). WSMO Working Draft. (2006) http://www.wsmo.org/TR/d2/v1.3/

[30] G. Schreiber, H. Akkermans, A. Anjewierden, R. De Hoog, N. Shadbolt, W. v. De Velde, B. Wielinga. Knowledge Engineering and Management: the CommonKADS Methodology. Mit Press. (1999)

[31] D. Fensel, E. Motta, F. Van Harmelen, V. R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, B. Wielinga. The Unified Problem-Solving Method Development Language UPML. Knowledge and Information Systems. 5, 83--131 (2003)

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

[33] Domingue, J., Cabral, L., Hakimpour, F., Sell, D. and Motta, E.: IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. In Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany. CEUR Workshop Proceedings, Vol. 113 (2004)

[34] WSMO Working Group. Deliverable D2v1.2 Web Service Modeling Ontology (WSMO) (2005). http://www.wsmo.org/TR/d2/v1.2

[35] L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, V. Tanasescu, C. Pedrinaci, B. Norton, IRS-III: A Broker for Semantic Web Services based Applications. In proceedings of the 5 th International Semantic Web Conference (ISWC 2006)

[36] Cabral, L., Domingue, J., Motta, E., Payne, T. and Hakimpour, F. (2004). Approaches to Semantic Web Services: An Overview and Comparisons. In proceedings of the First Euro-pean Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece. LNCS 3053

[37] G. Kapitsaki, D.A. Kateros, I.E. Foukarakis, G. N. Prezerakos, D.I. Kaklamani and I.S. Venieris, Service Composition: State of the art and future challenges, http://www.ist-sms.org/Documents/C8.pdf

[38] D. Berardi et al., “Reasoning about Actions for e-Service Composition”, ICAPS Workshop on Planning for Web Services (P4WS 2003), 2003

[39] S. McIlraith and T.C. Son. Adapting golog for composition of semantic web services. In Proc. Of KR-02, pages 482{496, 2002. Morgan Kaufmann.

[40] K. Erol, J. Hendler and D. S. Nau, “Semantics for Hierarchical Task Network Planning”, UMIACS Technical Report, 1994.

[41] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia, “Automatic Web Services Composition Using SHOP2”, Workshop on Planning for Web Services, 2003.

[42] A. Patil, S. Oundhakar, A. Sheth, K. Verma, “METEOR-S Web Service Annotation Framework”, The Proceedings of the Thirteenth International World Wide Web Conference, May 2004 (WWW2004)

[43] Semantic Annotations for Web Services Description Language, http://www.w3.org/2002/ws/sawsdl/

[44] NetBeans.org. (2007, 11.05.2010). NetBeans IDE 6.1 SOA Pack Documentation. Available: http://netbeans.org/kb/61/soa/index.html

[45] JOpera. (2010, 12.05.2010). JOpera for Eclipse. Available: http://www.jopera.org

[46] F. Benedikt, et al. (2008, 02.02.2010). Hybrid OWL-S Web Service Matchmaker. Available: http://www-ags.dfki.uni-sb.de/~klusch/i2s/SMR2-2009-owlsmx3.pdf

[47] H. Li, et al., "Automatic Composition of Web Services Based on Rules and Meta-Services," presented at the 11th International Conference on CSCW in Design, Melbourne, Australia, 2007.

[48] S. Narayanan and S. A. McIlraith, "Simulation Verification and Automated Composition of Web Service," presented at the 11th International World wide Web conference, Hawaii, USA 2002.

[49] I. Paik and D. Maruyama, "Automatic Web Services Composition Using Combining HTN and CSP," presented at the 7th IEEE International Conference on Computer and Information Technology (CIT 2007), Aizu-Wakamatsu City, Fukushima, Japan, 2007.

[50] X. Li and C. Wu, "Research on OWLS Service Automatic Composition Based on Planning," Wuhan, China 2009.

[51] W3C. 09.05.2010). OWL-S: Semantic Markup for Web Services. Available: http://www.w3.org/Submission/OWLS/

[52] LSDIS. (2005, 06.03.2010). METEOR-S: Semantic Web Services and Processes. Available: http://lsdis.cs.uga.edu/projects/meteor

[53] C. Feier, D. Roman, A. Polleres, J. Domingue, M. Stollberg, and D. Fensel, Towards Intelligent Web Services: The Web Service Modeling Ontology (WSMO), International Conference on Intelligent Computing (ICIC) (2005) http://axel.deri.ie/publications/feie-etal-2005.pdf

[54] Berners-Lee, T., Hendler, J., Lassila, O.: The SemanticWeb. Scientific American 284 (2001)

[55] Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and abstract syntax. Recommendation 10 February 2004, W3C (2004)

[56] Brickley, D., Guha, R.V., eds.: RDF Vocabulary Description Language 1.0: RDF Schema. (2004) W3C Recommendation 10 February 2004.

[57] Dean, M., Schreiber, G., eds.: OWL Web Ontology Language Reference. (2004) W3C Recommendation 10 February 2004.

[58] WSML working group: WSML homepage (since 2004) http://www.wsmo.org/wsml/

[59] McIlraith, S., Son, T.C., Zeng, H.: SemanticWeb Services. IEEE Intelligent Systems, Special Issue on the Semantic Web 16 (2001) 46–53

[60] Martin, D., ed.: OWL-S 1.1 Release. (2004) http://www.daml.org/services/owl-s/1.1/

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

[62] WSMO working group: WSMO homepage (since 2004) http://www.wsmo.org/

[63] Fensel, D., Bussler, C., Ding, Y., Omelayenko, B.: The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications 1 (2002)

[64] WSMX working group: WSMX homepage (since 2004) http://www.wsmx.org/

[65] The OWL-S Coalition. OWL-S 1.1 draft release, 2004. http://www.daml.org/services/owl-s/1.1/

[66] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN planning for web service composition using SHOP2. Journal of Web Semantics, 1(4):377-396, 2004.

[67] R. Reiter. Knowledge in Action. MIT Press, 2001.

[68] H.J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R.B. Scherl. Golog: a logic programming language for dynamic domains. Journal of Logic Programming, 31(1-3):59-83, 1997.

[69] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. JAIR, 20:379-404, 2003.

[70] M. Thielscher. Introduction to the Fluent Calculus. ETAI, 2(3{4):179{192, 1998.

[71] F. Baader , C. Lutz , M. Miličić , U. Sattler , F. Wolter, A description logic based approach to reasoning about web services, In Proceedings of the WWW 2005 Workshop on Web Service Semantics (WSS2005) http://www.informatik.uni-bremen.de/~clu/papers/archive/wss05.pdf

[72] Michael Stollberg , Cristina Feier , Dumitru Roman , Dieter Fensel, Semantic Web Services – Conceprts and Technology, Language Technology, Ontologies, and the Semantic Web (2006) http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.87.5391

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

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

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

[76] D. Roman, H. Lausen, and U. Keller (eds.). Web Service Modeling Ontology (WSMO). Deliverable D2, final version 1.2, 2005. Available from http://www.wsmo.org/TR/d2/

[77] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - A Semantic Service-Oriented Architecture. In Proceedings of the International Conference on Web Service (ICWS 2005), 2005.

[78] J. de Bruijn (ed.). The Web Service Modeling Language WSML. WSML Deliverable D16.1 _nal version 0.2, 2005. available from http://www.wsmo.org/TR/d16/d16.1/v0.2/

[79] OMG: The Object Management Group. Meta-object facility, version 1.4, 2002.

[80] S. Staab and R. Studer, editors. Handbook on Ontologies in Information Systems. International Handbooks on Information Systems. Springer, 2004.

[81] D. Fensel. Ontologies: A Silver Bullet for Knowledge Management and ECommerce. Springer, Berlin, Heidelberg, 2 edition, 2003.

[82] G. Wiederhold. Mediators in the architecture of the future information systems. Computer, 25(3):38{49, 1994.

[83] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In Proceedings of the First International Semantic Web Conference, Springer, 2002

[84] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In Proceedings of the 12th International Conference on the World Wide Web, Budapest, Hungary., 2003.

[85] U. Keller, R. Lara, and A. Polleres (eds.). WSMO Web Service Discovery. Deliverable D5.1, 2004. available at: http://www.wsmo.org/TR/d5/d5.1/

[86] S. Grimm, B. Motik, and C. Preist. Variance in e-business service discovery. In Proc. of the ISWC 2004 workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan, Nov. 2004, 2004.

[87] M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In Proc. of the ISWC 2004 workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan, Nov. 2004, 2004.

[88] M. Stollberg, U. Keller, and D. Fensel. Partner and Service Discovery for Collaboration Establishment on the SemanticWeb. In Proceedings of the Third International Conference on Web Services, Orlando, Florida, 2005.

[89] J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web. In Proceedings of the 14th International World Wide Web Conference (WWW2005), Chiba, Japan, 2005., 2004. http://www2005.org/cdrom/docs/p623.pdf

[90] J. Cardoso and A. Sheth, "Introduction to Semantic Web Services and Web Process Composition," in Semantic Web Process: powering next generation of processes with Semantics and Web Services, Lecture Notes in Computer Science, Springer, 2005

[91] SWSI, Semantic Web Services Initiative (SWSI). http://www.swsi.org/, 2004.

[92] OWL-S, OWL-based Web Service Ontology. http://www.daml.org/services/owl-s/, 2004.

[93] WSMO, Web Services Modeling Ontology (WSMO). http://www.wsmo.org/, 2004.

[94] WSML, Web Service Modeling Language (WSML). http://www.wsmo.org/wsml/index.html, 2004.

[95] WSMX, Web Services Execution Environment (WSMX). http://www.wsmx.org/, 2004.

[96] LSDIS, METEOR-S: Semantic Web Services and Processes. http://lsdis.cs.uga.edu/Projects/METEOR-S/index.php, 2004.

[97] SWWS, Semantic Web Enabled Web Service. http://swws.semanticweb.org/, 2004, Digital Enterprise Research Institute (DERI).

[98] DERI, Digital Enterprise Research Institute (DERI). http://www.deri.ie/, 2004.

[99] Michael C. Jaeger , Gregor Rojec-goldmann , Christoph Liebetruth , Gero Mühl , Kurt Geihs, Ranked Matching for Service Descriptions using OWL-S, In KiVS 2005, Informatik Aktuell (2005

[100] Marja-Riitta Koivunen and Eric Miller. W3c semantic web activity. In Semantic Web Kick- Off in Finland, pages 27–44, November 2001. http://www.w3.org/2001/12/semweb-fin/w3csw

[101] Anupriya Ankolenkar et al. Daml-s: A semantic markup language for web services. In Proceedings of 1st Semantic Web Working Symposium (SWWS' 01), pages 441–430, Stanford, USA, August 2001. Stanford University.

[102] Frank Manola et al. RDF Primer. Technical report, W3C, http://www.w3.org/TR/rdf-primer/ 2004.

[103] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language overview. Technical report, W3C, http://www.w3.org/TR/owl-features/ 2004.

[104] Sheila A. McIlraith and David L. Martin. Bringing semantics to web services. IEEE Intelligent Systems, 18:90–93, January/February 2003.

[105] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. Technical report, http://www.daml.org/services/ 2004.

[106] Jos De Bruijn, Dieter Fensel, Uwe Keller, Ruben Lara, Using the Web Service Modelling Ontology to enable Semantic eBusiness, Volume: 48, Issue: 12, Publisher: ACM, Pages: 43 Communications of the ACM, (2005)

[107] Robert Allen and David Garlan. A Formal Basis for Architectural Connection. ACM Transactions on Software Engineering and Methodology, 6(3):213{249, July 1997.

[108] Dieter Fensel and Christoph Bussler. The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications, 1(2):113137, 2002.

[109] Ruben Lara, Axel Polleres, Holger Lausen, Dumitru Roman, Jos de Bruijn, and Dieter Fensel. A Conceptual Comparison between WSMO and OWL-S. Final Draft D4.1v0.1, WSMO, 2005. http://www.wsmo.org/TR/d4/d4.1/v0.1/

[110] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, Katia Sycara, OWL Web Ontology Language for Services (OWL-S). W3C Member Submission, 22 November 2004. http://www.w3.org/Submission/OWL-S/.

[111] World Wide Web Consortium. Semantic Web Services Interest Group. http://www.w3.org/2002/ws/swsig/ (2003)

[112] Intelligent Software Agents Lab in the Robotics Institute at Carnegie Mellon I University. The Intelligent Software Agents Lab. http://www-2.cs.cmu.edu/~softagents/index.html (2001)

[113] Christopher Walton, Agency and the Semantic Web, Oxford University Press, 2007.

[114] Milanovic, N., & Malek, M. (2004). Current solutions for Web service composition. IEEE Internet Computing, 8(6), 51 - 59

[115] Beek, M. H. t., Bucchiarone, A., & Gnesi, S. (2006). A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods: Technical Report 2006-TR-15, ISTI, Consiglio Nazionale delle Ricerche.

[116] Ralph W. Feenstra, Marijn Janssen, and René W. Wagenaar. Evaluating Web Service Composition Methods: the Need for Including Multi-Actor Elements, Electronic Journal of e-Government Volume 5 Issue 2 2007 (153-164)

[117] Drew V. McDermott, "Estimated-Regression Planning for Interactions with Web Services", in Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, 2002, pp. 204-211.

[118] Ugur Kuter, Dana Nau, Marco Pistore and Paolo Traverso, "A Hierarchical Task-Network Planner based on Symbolic Model Checking", in Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling, 2005, pp. 300-309.

[119] Jianhong Zhang, Shensheng Zhang, Jian Cao and Yujie Mou., "Improved HTN Planning Approach for Service Composition", in Proceedings of the 2004 IEEE International Conference on Service Computing, 2004, pp. 609-612.

[120] Chan, K. S. M., Bishop, J., and Baresi, L. (2007). Survey and comparison of planning techniques for web service composition. Technical report, Dept Computer Science, University of Pretoria.

Annex 1 – pSHIELD Glossary

Concept	Description
Application Processor (AP)	Main processing unit
Atomic pSHIELD SPD Component	Is a generic atomic SPD Functionality (innovative or legacy) provided by a pSHIELD device at node, network or middleware level.
Audit	Involves recognizing, recording, storing, and analyzing information related to SPD relevant activities. The resulting audit records can be examined to determine which SPD relevant activities took place.
Authorized User	A user who possesses the rights and/or privileges necessary to perform an operation
Automatic Web Service Composition and Interoperation	This task involves the automatic selection, composition, and interoperation of Web services to perform some complex task, given a high-level description of an objective.
Automatic Web Service Discovery	Is an automated process for location of Web services that can provide a particular class of service capabilities, while adhering to some client-specified constraints.
Automatic Web Service Invocation	Is the automatic invocation of an Web service by a computer program or agent, given only a declarative description of that service, as opposed to when the agent has been pre-programmed to be able to call that particular service.
Availability	Readiness for correct service. The correct service is defined as delivered system behaviour that is within the error tolerance boundary.
Awareness	Capability of the Cognitive Radio to understand, learn, and predict what is happening in the radio spectrum, e.g., to identify the transmitted waveform, to localize the radio sources, etc.
Bridge	Is a network device that is at the link layer of the ISO / OSI model and translates from one physical media to another within the same local network.
Class and Family	The CC has organized the components into hierarchical structures: Classes consisting of Families consisting of components. This organization into a hierarchy of class - family - component - element is provided to assist consumers, developers and evaluators in locating specific components
Cognitive Radio	Is an intelligent wireless communication system that is aware of its surrounding environment (i.e., outside world), and uses the methodology of understanding-by-building to learn from the environment and to adapt its internal states to statistical variations in the incoming Radio-Frequency (RF) stimuli by making corresponding changes in certain operating parameters (e.g., transmit-power, carrier-frequency, and modulation strategy) in real-time, with two primary objectives in mind: (i) highly Reliable and Dependable communications whenever and wherever needed and (ii) efficient utilization of the radio spectrum.
Common Criteria	The Common Criteria for Information Technology Security Evaluation (abbreviated as Common Criteria or CC) is an international standard (ISO/IEC 15408) for computer security certification. It is currently in version 3.1. Common Criteria is a framework in which computer system users can specify their security functional and assurance requirements, vendors can then implement and/or make claims about the security attributes of their products, and testing laboratories can evaluate the products to determine if they actually meet the claims. In other words, Common Criteria provides assurance that the process of specification, implementation and evaluation of a computer security product has been conducted in a rigorous and standard manner.
Common Criteria Approach	Is approach based in three fundamental part: • Assets to protect and in particular SPD attribute of these assets definition

Concept	Description
	• Threats identifications (Fault Errors Failures); in our approach faults are grouped in HMF (FUA, NFUA) and NHMF. • SPD functionalities (whose purpose is to mitigate threats) are implemented to meet pSHIELD SPD objectives.
Composability	Is the possibility to compose different (possibly heterogeneous) SPD functionalities (also referred to as SPD components) aiming at achieving in the considered system of Embedded System Devices a target SPD level which satisfies the requirements of the considered scenario.
Confidentiality	Property that data or information are not made available to unauthorized persons or processes.
Contiki Operating System	Contiki is also an open source, highly portable, multi-tasking operating system for memory-efficient networked ESDs and WSNs
Control Algorithms	Retrieves the aggregated information on the current SPD status of the subsystem, as well as of the other interconnected subsystems, by the pS-CA interface connected to the Semantic Knowledge Representation; such retrieved information is used as input for the Control Algorithms. The outputs of the Control Algorithms consist in decisions to be enforced in the various ESDs included in the pSHIELD subsystem controlled by the Security Agent in question; these decisions are sent back via the pS-MS interface, as well as communicated to the other Security Agents on the Overlay, through the pS-OS interface.
Control system	A system that uses a regulator to control its behaviour
Core SPD Services	The core SPD services are a set of mandatory basic SPD functionalities provided by a pSHIELD Middleware Adapter in terms of pSHIELD enabling middleware services. The core SPD services aim to provide a SPD middleware environment to actuate the decisions taken by the pSHIELD Overlay and to monitor the Node, Network and Middleware SPD functionalities of the Embedded System Devices under the pSHIELD Middleware Adapter control.
Correct service	Delivered system behaviour is within the error tolerance boundary.
Cryptographic Algorithms	Algorithms to hiding the information, to provide security and information protection against different forms of attacks
Dependability	Is a composite concept that encompasses the following attributes: Availability, Reliability, Safety Integrity, Maintainability
Desired objectives	Desired objectives normally specified by the user.
Desired service	Delivered system behavior is at or close to the desired trajectory.
Discovery	Provide to the pSHIELD Middleware Adapter the information, raw data, description of available hardware resources and services in order to allow the system composability
Discovery Engine	it is in charge to handle the queries to search for available pSHIELD components sent by the Composition service.
Discovery Protocol	it is in charge to securely discover all the available SPD components description stored in the Service Registry, using a specific protocol
Disturbance	Anything that tends to push system behavior off the track is considered a disturbance. A disturbance can occur within a system or from the external environment.
Error	Deviation of system behavior/output from the desired trajectory.
Error tolerance boundary	A range within which the error is considered acceptable by a system or user. This boundary is normally specified by the user.
Evaluator	An independent person involved in the judgment about the measure of the SPD functions.
Failure	An event that occurs when system output deviates from the desired service and is beyond the error tolerance boundary.
Fault	Normally the hypothesized cause of an error is called fault. It can be internal or external to a system. An error is defined as the part of the total state of a system that may lead to subsequent service failure. Observing that many errors do not reach a system's external state and

Concept	Description
	cause a failure, Avizienis et al. have defined active faults that lead to error and dormant faults that are not manifested externally.
Fault injection	This block has the responsibility to inject a fault into Demodulator
Faults with Unauthorized Access	The class of Faults with unauthorized access (FUA) attempts to cover traditional security issues caused by malicious attempt faults. Malicious attempt fault has the objective of damaging a system. A fault is produced when this attempt is combined with other system faults.
Feedback	Use of the information observed from a system's behavior to readjust/regulate the corrective action/control so that the system can achieve the desired objectives.
Feedback control system	A control system that deploys a feedback mechanism. This is also called a closed-loop control system.
Filter Engine	It is in charge to semantically match the query with the descriptions of the discovered SPD components
Flash Memory	It stores the bit-stream and system status information
Forecasting (Fault)	Mechanism that predicts faults so that they can be removed or their effects can be circumvented
Functional Component	Describes a functional entity that, in general, does not have necessarily a physical counterpart (e.g. a software functionality, a middleware service, an abstract object, etc.).
Gateway	Is a network device that operates at the network layer and above the ISO / OSI model. Its main purpose is to transmit network packets outside a local area network (LAN).Functional Component
Grounding	Provides details on how to interoperate with a service, via messages.
Health Status Monitoring (HSM)	Monitoring for checking the status of each individual component.
Hub	Is a concentrator, a network device that acts as a routing node of a data communications network
Human-Made Faults	Human-made faults result from human actions. They include absence of actions when actions should be performed (i.e., omission faults). Performing wrong actions leads to commission faults. HMF are categorized into two basic classes: faults with unauthorized access (FUA), and other faults (NFUA).
Hybrid Automata	Is composed by automaton formulation hybrid formulation that Permit to choose the most suitable configuration rules for components that must be composed on the basis of the Overlay control algorithms.
HYDRA Middleware	Middleware for Heterogeneous Physical Devices
I/O Interface	(I/O) to connect to any peripheral and to the rest of the pSHIELD embedded functionalities.
Innovative SPD Functionalities	Reside in the pSHIELD Middleware, Network and Node Adapters. They are constituted by a variety of pSHIELD-specific components. Each pSHIELD-specific component. represents an innovative SPD functionality ad hoc developed for the pSHIELD project which is included in the pSHIELD Node, Network or Middleware Adapter.
Integrity	Absence of malicious external disturbance that makes a system output off its desired service
Legacy Device Component	i.e. the SPD functionalities already present in the legacy devices which can be accessed through the pSHIELD Adapters; they can be classified in Node, Network and Middleware Component according to whether they are included in a legacy Node/Network/Middleware which can be accessed through the corresponding pSHIELD Adapter.
Legacy Embedded System Device (L-ESD)	It represents an individual, atomic physical Embedded System device characterized by legacy Node, Network and Middleware functionalities.
Legacy Functionalities of L-ESD	Is a functionality partitioned into three subsets: - Node layer functionalities: hardware functionalities such as processors, memory, battery, I/O interfaces, peripherals, etc. - Network layer functionalities: communication functionalities such as connectivity, protocols stack, etc.

Concept	Description
	- Middleware layer functionalities: firmware and software functionalities such as services, functionalities, interfaces, protocols, etc.
Legacy Middleware Services	Includes all the legacy middleware services (i.e. messaging, remote procedure calls, objects/content requests, etc.) provided by the Legacy Embedded System Device which are not pSHIELD-compliant.
Legacy Network Services	Includes all the legacy network services (protocol stacks, routing, scheduling, Quality of Service, admission control, traffic shaping, etc.) provided by the Legacy Embedded System Device which are not pSHIELD-compliant.
Legacy Node Capabilities	Component includes all the legacy node capabilities (i.e. battery, CPU, memory, I/O ports, IRQ, etc.) provided by the Legacy Embedded System Device which are not pSHIELD compliant.
Life-Cycle support elements	It is the set of elements that support the aspect of establishing discipline and control in the system refinement processes during its development and maintenance. In the system life-cycle it is distinguished whether it is under the responsibility of the developer or the user rather than whether it is located in the development or user environment. The point of transition is the moment where the system is handed over to the user.
Maintainability	Ability to undergo modifications and repairs.
Mean	All the mechanisms that break the chains of errors and thereby increase the dependability of a system
Memory	Memory RAM, SRAM, DRAM,
Metadata	Information that describe set of data
Micro/Personal nodes	Nodes that are richer than Nano Nodes in terms of hardware and software resources, network access capabilities, mobility, interfaces, sensing capabilities, etc.
Middleware Layer	Includes the software functionalities that enable the discovery, composition and execution of the basic services necessary to guarantee SPD as well as to perform the tasks assigned to the system (for example, in the railway scenario, the monitoring functionality)
Nano Nodes	Are typically small ESD with limited hardware and software resources, such as wireless sensors.
Net Device	Components used to connect computers or other electronic devices
Network CAN	Control Area Network
Network LAN	Local Area Network
Network Layer	Includes the communication technologies (specific for the rail transportation scenarios) that allow the data exchange among pSHIELD components, as well as the external world. These communication technologies, as well as the networks to which pSHIELD is interconnected can be (and usually are) heterogeneous.
Network MAN	Metropolitan Area Network
Network VPN	Virtual Private Network
Network WAN	Wide Area Network
Node Layer	Includes the hardware components that constitute the physical part of the system.
Node Metrics / Health Status	It receives periodic health messages and metrics from the other blocks. This block contains the information about the full node configuration, metrics and health status. If e.g. the “Assertions” block detects some erroneous output, “Node Metrics / Health Status” block receives this information and must act accordingly.
NonHuman-Made Faults	NHMF refers to faults caused by natural phenomena without human participation. These are physical faults caused by a system’s internal natural processes (e.g., physical deterioration of cables or circuitry), or by external natural processes. They can also be caused by natural phenomena
Non-Volatile Memory (NVM)	Memory ROM, EEPROM, FLASH, Hard Disk
Not Faults with	Human-made faults that do not belong to FUA. Most of such faults are

Concept	Description
Unauthorized Access	introduced by error, such as configuration problems, incompetence issues, accidents, and so on.
Open-loop control system	A control system without a feedback mechanism.
Overlay	Consists of a set of SPD Security Agents, each one controlling a given pSHIELD subsystem.
Overlay Layer	The "embedded intelligence" that drives the composition of the pSHIELD components in order to meet the desired level of SPD. This is a software layer as well.
Physical Component	Describes an entity that can be mapped into a physical object (e.g. a hardware component).
Plant	A system that is represented as a function P(.) that takes its input as a functional argument
Power Management (PM)	Module for managing power sources, monitoring power consumption, etc.
Power nodes	Is an ES node that offers high performance computing in one self-contained board offering data storage, networking, memory and (multi-)processing.
Prevention (Fault)	Mechanism that deals with preventing faults incorporated into a system
Privacy	The right of an entity (normally a person), acting in its own behalf, to determine the degree to which it will interact with its environment, including the degree to which the entity is willing to share information about itself with others.
pSHIELD Adapter	Is a technology dependent component in charge of interfacing with the legacy Node, Network and Middleware functionalities (through the MS, NS and NC interfaces). The legacy functionalities can be enhanced by the pSHIELD Adapter in order to make them pSHIELD-compliant, i.e. they become SDP legacy device components. In addition, the pSHIELD Adapter includes Innovative SPD functionalities which are SPD pSHIELD-specific components, which can be composed with other SPD components. The pSHIELD Adapter exposes the technology independent pSHIELD Middleware layer functionalities that are used by the Security Agent component.
pSHIELD Communication	This block interfaces SPD Node to pShield Network. It is composed by: Ethernet interface: it allows a communication data interface based on a TCP/IP protocol Message encoding/decoding: it receives the commands from pShield network, decodes them, and acts accordingly. It also sends messages to the network.
pSHIELD Demonstrator	Demonstrator for the project that Have the task To monitor on-carriage environment; To integrate the sensors at the wagon with the M2M platform; To allow secure interoperability of sensor information (between railway infrastructure and third party service provider).
pSHIELD Embedded System Device (pS-ESD)	It is a L-ESD equipped at least with the minimal set of pSHIELD functionalities at Middleware Layer. The pS-ESD exposes the same functionalities as the L-ESD plus an additional interface: the pSHIELD Middleware layer services.
pSHIELD Middleware Adapter	Is a component partitioned in the Core SPD services and in the Innovative SPD functionalities. These two components are linked through the pS-MS interface. The pSHIELD Middleware Adapter should also carry into operation the decisions taken by the Overlay and communicated via the pS-MS interface by actually composing the discovered SPD functionalities. The pSHIELD Middleware Adapter includes a set of Innovative SPD functionalities interoperating with the legacy ESD middleware services (through the MS interface) in order to make them discoverable and composable SPD functionalities.
pSHIELD Multi-Layered Approach	The pSHIELD multi-layered approach considers the partition of a given Embedded System into three

Concept	Description
	technology-dependent horizontal layers: the node layer (meaning the hardware functionalities), the network layer (meaning the communication functionalities) and the middleware layer (meaning thesoftware functionalities).
pSHIELD Network Adapter	Includes a set of Innovative SPD functionalities interoperating with the legacy ESD network services (through the NS interface) and the pSHIELD Node Adapter (through the pS-NC interface) in order to enhance them with the pSHIELD Network layer SPD enabling technologies (such as Smart Transmission). This adapter is also in charge to provide (through the pS-NS interface) all the needed information to the pSHIELD Middleware adapter to enable the SPD composability of the Network layer legacy and Network pSHIELD-specific functionalities. Moreover, the pSHIELD Network Adapter translates the technology independent commands, configurations and decisions coming from the pS-NS interface into technology dependent ones and enforce them also to the legacy Network functionalities through the NS interface.
pSHIELD Node	Is an Embedded System Device (ESD) equipped with several legacy Node Capabilities and with a pSHIELD Node Adapter. A pSHIELD Node is deployed as a hardware/software platform, encompassing intrinsic, Innovative SPD functionalities, providing proper services to the other pSHIELD Network and Middleware Adapters to enable the pSHIELD Composability and consequently the desiredsystem SPD
pSHIELD Node Adapter	Includes a set of Innovative SPD functionalities interoperating with the legacy ESD node capabilities (using the NC interface) in order to enhance them with the pSHIELD Node layer SPD enabling technologies (such as FPGA Firmware and Lightweight Asymmetric Cryptography). This adapter is in charge to provide (through the pS-NC interface) all the needed information to the pSHIELD Middleware adapter to enable the SPD composability of the Node layer legacy and Node pSHIELD-specific functionalities. Moreover, the pSHIELD Node Adapter translates the technology independent commands, configurations and decisions coming from the pS-NC interface into technology dependent ones and enforce them also to the legacy Node functionalities through the NC interface.
pSHIELD Proxy (pS-P)	Is a technology dependent component of a pS-SPD-ESD that, interacting with the available legacy Node, Network and Middleware capabilities and functionalities (through the NC, NS and MS interfaces, respectively), provides all the needed pSHIELD enhanced SPD functionalities.
pSHIELD SPD Embedded System Device (pS-SPD-ESD):	It is a pS-ESD equipped at least with the minimal set of pSHIELD Overlay functionalities. The pS-SPD-ESD exposes the same functionalities as the pS-ESD plus an additional interface: the pSHIELD Overlay layer SPD services provided by a so-called Service Agent operating in that ESD.
pSHIELD Subsystem (pS-S)	Is an architecture of a set of Embedded System Devices including several L-ESD, connected to several pS-ESD and one and only one pS-SPD-ESD. Connections between two generic ESDs (L-ESD, pS-ESD or pS-SPD-ESD) can be performed, by means of legacy functionalities at Node, Network and/or Middleware layer, through the so-called NC, NS and MSfunctionalities, respectively.
pSHIELD-Specific Components	It is i.e. the innovative SPD functionalities ad hoc developed for the pSHIELD project which are included in the pSHIELD Adapters. They can be classified in Node Network and Middleware pSHIELD-specific components according to whether they are included in the pSHIELD

Concept	Description
	Node, Network or Middleware Adapter. They can be directly accessed by pSHIELD Middleware Core SPD Services through the pSNC, pS-NS and pS-MS interfaces.
Query Preprocessor	It is in charge to enrich the query sent by the Composition service with semantic information related to the peculiar context.
Reconfigurability	Provide self-configuration of some internal parameters according to the observed radio spectrum.
Reconfiguration / Recovery	This block runs at the PPC static core. It must receive periodically health status information, otherwise it restarts the system
Reconfiguration/Recovery Controller	This is a hard processor or microcontroller, responsible for either reconfiguring the node or recovering in case of an error.
Recovery Watchdog Timer (RWDT)	Timer for restarting recovery if no activity is detected from the SHSM.
Regulator	A control function whose role is to regulate the input of the plant, under the influence of the environment such as instructions of the user, observed error, input disturbance, etc. to achieve the desired objective.
Reliability	Continuity of correct service even under a disturbance.
Removal (Fault)	mechanism that permits to the system to record failures and remove them via a maintenance cycle
Repeater	A digital device that amplifies, reshapes, retimes, or performs a combination of any of these functions on a digital input signal for retransmission
Router	Is a device that forwards data packets between telecommunications networks, creating an overlay internetwork. A router is connected to two or more data lines from different networks.
Rules for Discovery, Configuration and Composition of the SPD Components.	Design and implementation of the Control Algorithms which, on the basis of the sensed metadata (i.e) on the basis of the ontological description (possibly semantically enriched) of the SPD Components provide Rules for discovery, configuration and composition of the SPD components.
Safety	Absence of catastrophic consequences on the users and the environment.
Service failure	An event that occurs when system output deviates from the desired service and is beyond the error tolerance boundary.
SDP Network	Is a Network implementable and interoperable with standard networks to comply the main business cases of the application scenarios.
Seamless Approach	Common approach which leaves out of consideration the specificity of the underlying technologies providing enriched SPD functionalities to heterogeneous Embedded Systems
Secure Service Discovery	Allows any pSHIELD Middleware Adapter to discover in a secure manner the available SPD functionalities and services over heterogeneous environment, networks and technologies that are achievable by the pSHIELD Embedded System Device (pS-ESD) where it is running.
Secure/Privacy (SP)	Module to perform security and privacy actions, such as encryption, decryption, key generation, etc.
Security	Is a composite of the attributes of confidentiality, integrity (in the security context, “improper” means “unauthorized"), and availability (for authorized actions only),
Security Agent	Is a technology-independent component in charge of aggregating the information coming from the pSHIELD Middleware Services provided by the internal pSHIELD Adapter or by other pSHIELD Proxies located in the same subsystem. The Security Agent

Concept	Description
	is also in charge of gathering the information coming from other Security Agents connected on the same Overlay (through the pS-OS interface). The Security Agent includes proper control algorithms working on the basis of the available information; the decisions taken by these Control Algorithms are enforced through the pS-MS and the pS--OS interfaces.
Semantic Database	It holds any semantic information related to the pSHIELD components (interface, contract, SPD status, context, etc.).
Semantic Engine (Reasoner)	Enable interoperability within Middleware Layer and rule based discovery and composition within Overlay Agents.
Semantic Knowledge Repository	Is (i.e. a database) that storing the dynamic, semantic, enriched, ontological aggregated representation of the SPD functionalities of the pSHIELD subsystem controlled by the SPD Security Agent;
Semantic Knowledge Representation	Is in charge of bi-directionally exchanging technology independent (and semantic enriched) information from the pS-MS and the pS-OS interfaces. It is also in charge to provide such information via the pS-SKR interface to the Control Algorithms component.
Semantic Model	It is a conceptual model in which semantic information is included.
Sensor/Actuator	Are represented by the Core SPD Services lying at the pSHIELD Middleware layer.
Sensors/Actuators	Represent the Core SPD Services lying at the pSHIELD Middleware layer.
Service Composition	Is in charge to select atomic SPD services that, once composed, provide a complex and integrated SPD functionality that is essential to guarantee the required SPD level. The service composition is a pSHIELD Middleware Adapter functionality that cooperates with the pSHIELD Overlay in order to apply the configuration strategy decided by the Control Algorithms residing in the pSHIELD Security Agent.
Service Grounding	Specifies the details of how an agent can access a service-details having mainly to do with protocol and message formats, serialization, transport, and addressing
Service Orchestration	Deploy, execute and monitoring SPD services.
Service Profile	Tells "what the service does", in a way that is suitable for a service-seeking agent (or matchmaking agent acting on behalf of a service-seeking agent) to determine whether the service meets its needs.
Services Model	Tells a client how to use the service, by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes.
Services Registry	It acts as a database to store the service entries
Smart SPD Driven Transmission	New advances signal processing techniques.
SOA	Service-oriented architecture
Software Agent	Permit a computer-interpretable description of the service.
Software Defined Radio (SDR)	Software programmable Components
Software Wireless Sensor Networks (WSN)	Software part that can be layered into three levels: sensor software, node software and sensor network software.
SPD	Security Privacy Dependability
SPD Component	Is defined as a functionality which (i) offers a given SPD service through an interface which can be semantically described according to the provided SPD Metrics, (ii) can be accessed through the pSHIELD Middleware Core SPD Services for being configured (if

Concept	Description
	necessary) and activated (or deactivated).
SPD Metrics	Is the possibility to identify and quantify the SPD properties of each component, as well as the SPD properties of the overall system. SPD Metrics allow (i) users to define in an univocal way the requirements for the specific application, (ii) to describe the SPD level provided by the components, and (iii) to compute the SPD level achieved by the system through the Composability mechanism.
SPD Node	It is composed by the following sub- blocks: FM Signal Acquisition: this blocks principally handles the receiving of data samples from "FM Signal Generator" and pre-processes the data to feed to the "Demodulation Processing" block. This block provides also periodic status & metrics information to the "Node Metrics/Health Status" block. Demodulation Processing: it is responsible for the demodulation processing of the data coming from the "FM Signal Acquisition"
SPD Security Agent	Consists of two key elements: (i) the Semantic Knowledge Repository (i.e. a database) storing the dynamic, semantic, enriched, ontological aggregated representation of the SPD functionalities of the pSHIELD subsystem (ii) the Control Algorithms generating, on the grounds of the above representation, key SPD-relevant decisions (consisting, as far as the Composability feature is concerned, in a set of discovery, configuration and composition rules).
SPD Status	It represents the current SPD level associated to the function.
Special Purpose Processor	(SPP) module for any pre- or post-processing, such as compression/decompression, conversion, etc.
Persistant Storage	Used for storing the status of the system, a bit stream to program an FPGA, and/or the software for system start-up, operating system and application. it receives from each block check-pointing data. It is able to perform a stable write with this data (write on a circular buffer on flash memory, and then validate the just written data). On system restart, this module is able to recover the last valid data.
Switch	Is a computer networking device that connects network segments.
System	A composite constructed from functional components. The interaction of these components may exhibit new features/functions that none of the composite components possess individually.
System output	System behavior perceived by the user.
System Health Status Monitoring (SHSM)	Monitoring for checking the status of the whole system.
The Security/Privacy controller	Consists of one or more modules able to perform different security-related functionalities, such as Data Encryption, Data Decryption, Generation of Cryptographic Keys, etc
Threat	Include faults, errors and failures, as well as their causes, consequences and characteristics.
TinyOS	This operating system (OS) is a free and open source operating system and platform that is designed for WSNs.
Tolerance (Fault)	System architecture that deals with putting mechanisms in place that will allow a system to still deliver the required service in the presence of faults, although that service may be at a degraded level
User session	A period of interaction between users and SPD functional components.
WSDL	Web Services Description Language

Table 2 pSHIELD Glossary