Contract No. IST 2005-034891

Hydra

Networked Embedded System middleware for Heterogeneous physical devices in a distributed architecture

D12.8 Final Combined Internal Training Materials

Integrated Project
SO 2.5.3 Embedded systems

Project start date: 1st July 2006 **Duration: 48 months**

Published by the Hydra Consortium **28/02/2010 - version 1.0**
Coordinating Partner: Fraunhofer FIT

Project co-funded by the European Commission within the Sixth Framework Programme (2002 -2006)

Dissemination Level: Public

Document file: D12.8 Final Combined Internal Training Materials.doc

Work package: WP12 – Training

Task: T12.1 – Training

Document owner: University of Reading (UR)

Document history:

Version	Author(s)	Date	Changes made
0.1	Atta Badii, Junaid Raja Khan, Michael Crouch (UR)	03-02-2010	Initial TOC
0.2	Atta Badii, Junaid Raja Khan, Michael Crouch, Sebastian Zickau (UR)	15-02-2010	Additions to TOC; added content for chapters 3, 4 and 6; added chapter 7
0.5	Atta Badii, Junaid Raja Khan, Michael Crouch, Sebastian Zickau (UR)	18-02-2010	Core content collected together, and added to document.
0.7	Atta Badii, Junaid Raja Khan, Michael Crouch, Sebastian Zickau (UR)	25-02-2010	Preliminary version send to Partners for checking
0.9	Mads Ingstrup (UAAR), Francisco Milagro (TID), Sascha Effert (UoP), Amro-al-Akkad (FIT), Peter Kostelnik (TUK), Peter Kool (C-Net), Tobias Wahl (SIT)	26-02-2010	Added relevant sections on components and tool
1.0 - beta	Atta Badii, Junaid Raja Khan, Michael Crouch, Sebastian Zickau (UR)	03-03-2010	Added comments by all involved partners – ready for peer review
1.0 – pre final	Atta Badii, Junaid Raja Khan, Michael Crouch, Sebastian Zickau (UR)	05-03-2010	Added review comments
1.0	Atta Badii, Junaid Raja Khan, Michael Crouch, Sebastian Zickau (UR)	08-03-2010	Final Version

Internal review history:

Reviewed by	Date	Comments
Sascha Effert	03-03-2010	Accepted with comments
Trine F. Sørensen	03.03.2010	

Table of Content:

Figures:

1. Executive summary

This deliverable D12.8, titled "Final Combined Internal Training Materials" is intended to serve the researchers, technology experts, ambient intelligence solution developers and applications integrators who wish to examine the technologies used in Hydra and how they have been integrated to form the Hydra middleware platform plus the benefits that they could offer for solving specific design and requirement challenges.

The task of developing internal and external training materials has been an ongoing process during the Hydra project.

Internal training material has been in use by the Partners and people who are interested in the technical aspects of the project. Training materials have been subjected to periodic updates as informed by the feedback from the training sessions conducted to-date as well as revisions and new additions that have been added as they have become available throughout the course of the project. In particular the technical details re architectural and implementation issues have been updated continuously to reflect the latest developments and improvements as following the iterative cycles of design refinement and re-engineering as the project has evolved.

During the current phase the design and implementation decisions have been finalised and the technologies used within Hydra are now consolidated into what will be delivered as the final version of Hydra Platform at the end of this final period of the project.

Within its three sections; namely Hydra Technologies, Hydra Components, and Hydra Tools, this document covers the following five areas:

i) The underlying technologies used for the realisation of the components of the Hydra Platform

ii) The rationale for the design and implementation choices made in developing the Hydra platform

iii) The advantages/disadvantages of the adopted technologies, how these technologies support Hydra and any modifications made in adopting them, and what was accomplished.

iv) The various tools used in Hydra, what kind of usage they support and what technologies are used in these tools.

v) The description of the application domains as well as general concepts and technologies specifically related to Hydra.

Accordingly this document together with its sequel deliverable D12.9 "Final External Developer Workshop Materials" will provide both the high level architectural and sub-system level descriptions as well as the detail of the lower level functionality and APIs, etc. as used to build applications based on the Hydra Middleware Platform.

2. Introduction

2.1 Purpose, context and scope of this deliverable:

This deliverable D12.8, titled “Final Combined Internal Training Materials”, combines and finalises, as the title suggests, all 'internal' Hydra training into this single document which comprises three sections namely Hydra Technologies, Hydra Components, and Hydra Tools.

The aim of this deliverable is to highlight the use and promote understanding of technologies, tools and software components developed as part of the Hydra platform offered to a target audience comprising researchers and solution developers and integrators.

Accordingly this document, deliverable D12.8, summarises the descriptions of various Hydra components.

2.2 Structure of this document:

Following on from the Executive Summary and Introduction, an overview of the Hydra Research and Development and Technology Development (RTD) Objectives is given in Chapter 3, including an overview of the Hydra architecture highlighting the main design principles of the middleware.

Chapter 4 presents an overview of the technologies deployed in various Hydra components and also refers to some extensions which were made to improve these technologies within the scope of Hydra.

Chapter 5 sets out a detailed description of different components of Hydra and how the enabling technologies have been used to overcome certain design challenges and the added-value thus realised for the whole middleware platform.

In Chapter 6 explains Hydra tools and concepts and how they can be deployed when using Hydra to develop applications.

Chapter 7 presents a Summary and Chapters 8 provides a glossary with relevant Hydra terminology. Further reading suggestions and useful sources are given in the reference section in Chapter 9.

3. Hydra Objectives Overview

In the 21st Century, Embedded Systems can be found everywhere and can even be interconnected into networks consisting of many diverse devices forming the building blocks of the future Internet of Things. The Hydra project aims to alleviate the problems that European Industry is facing by researching and developing middleware for networked embedded systems. It will achieve this by allowing programmers to develop cost-effective, high-performance Ambient Intelligence (AmI) applications using heterogeneous physical devices. This has been facilitated by a series of development tools: The Hydra Software Development Kit (SDK), Device Development Kit (DDK) and the Integrated Development Environment (IDE).

The Hydra middleware is an intelligent software layer that is placed in between the operating system and its applications. Hydra contains several software components or managers, which have been carefully designed to handle the various tasks needed to support the cost-effective development of intelligent applications for networked embedded systems. Hydra can be incorporated in both new and existing networks of distributed devices, which operate with limited resources in terms of computing power, energy and memory usage. Hydra allows developers to incorporate heterogeneous physical devices into their applications by providing easy-to-use web service interfaces for controlling any type of physical device irrespective of its network interface technology.

Hydra is based on a semantic Model Driven Architecture for easy programming and incorporates solutions for device and service discovery, peer-to-peer communication and diagnostics. Hydra-enabled devices also offer secure and trustworthy communication through distributed security and social trust middleware components. Furthermore, Hydra specifically facilitates realisation of context-aware behaviour and management of data persistence on resource-constrained devices. Context-aware services are able to ubiquitously sense a user's environment and obtain information about the circumstances under which they are able to operate and thus adapt their behaviour in an intelligent way based on rules or stimuli. The Hydra SDK, DDK and IDE allows developers to create new, innovative networked embedded AmI applications and devices in a quick and cost-effective manner.

3.1 Technical objectives

The objectives of the HYDRA project are to research, develop and validate technologies and tools for building applications for networked embedded systems supporting ambient intelligence in a trusted and secure way.

The objectives of work to be performed thus consist of the following individual parts:

A. Development of a middleware based on a Service-oriented Architecture, to which the underlying communication layer is transparent, and consisting of:
   - Support for distributed as well as centralised ambient intelligent architectures
   - Support for reflective properties of components of the middleware
   - Support for security and trust enabling components

B. A generic semantic model-based architecture supporting model-driven development of applications

C. Tools (SDK, DDK, IDE) that will allow developers to develop applications on the middleware.

D. A business modelling framework for analysing the business sustainability of the developed applications.

E. Validation of the middleware, toolkit and business models in real end-user scenarios in three user domains:
   1. Facility management, smart homes
   2. Healthcare
   3. Agriculture

The project results consist of the following end products:

- HYDRA middleware for networked embedded systems

  Adds AmI applications to new and existing embedded systems and components
- HYDRA Software Development Kit (SDK), Device Development Kit (DDK) and IDE (Integrated Development Environment).

  Allows developers to rapidly create new networked embedded AmI applications and devices

The middleware will support cost-effective and innovative applications on embedded systems for new and already existing devices, which operate with limited resources in terms of computer power, energy and memory usage.

The SDK will allow developers to develop the innovative software applications with embedded ambient intelligence computing using the middleware, while the DDK will allow device developers to enable their devices to participate in a HYDRA network. The Hydra Middleware IDE will provide solutions supporting developers with high-level interfaces for developing networked embedded AmI applications.

An open source reference implementation will demonstrate the applicability and quality characteristics of the HYDRA middleware. The project will carry out foundational and component research and development as well as application and system integration within the following research areas:

- Embedded and mobile Service-oriented Architectures for ubiquitous networked devices
- Semantic Model-Driven Architecture for Ambient Intelligence implementation
- Ambient Intelligence support
- Hybrid architectures for Grid enabled networked embedded systems
- Wireless devices and networks with self-* properties (self-configuring, self-healing, etc.)
- Distributed social trust and behaviour as well as security and privacy
- Business innovation Embedded and mobile Service-orientated Architecture (SoA)

Every device, sensor, actuator using HYDRA middleware should be able to be considered as a unique service. Support for dynamic reconfiguration of devices, for example, will be included for self-configuration of AmI applications.

The HYDRA research and development challenges and objectives are:

- To address very small devices in terms of energy and computing power (scalability)
- To mask large differences in device resources and capabilities (heterogeneity)
- To handle intermittent connectivity (availability)
- To handle decentralised knowledge bases with no access to static server(s)
- To eliminate the need for pre-defined and persistent network topology
- To handle frequently changing network access by the user

3.1.1 Semantic MDA for AmI

The HYDRA middleware will have provision for creating AmI services and systems through a model-driven, semantic approach. The HYDRA middleware will integrate semantic web services with the device level and thus open up the interoperability of such systems. The HYDRA research and development challenges and objectives are:

- To define and engineer a uniform, open modelling language
- To define an open type of semantics and describe its function
- To make semantic Model Driven Architectures (MDA) operational in a distributed and possibly ad hoc world
- To provide tools for automatic ontology construction and discovery of devices
- To combine distributed information fractions to obtain a complete view

- To develop searchable libraries of models for different domains and devices.
- To enable communication between devices that might leave the network unpredictable

3.1.2 Ambient Intelligence support

The HYDRA middleware will apply relevant principles and methodologies from autonomic computing and apply them to decentralised network embedded systems. An essential problem in AmI is failure, both transient and permanent. There will be provision for monitoring systems, discovering failures, reasoning about failures, and reacting to failures within the HYDRA middleware. Due to the flexibility of the domain models future extensions into perception and cognition, for example, may be facilitated.

The HYDRA research and development challenges and objectives are:

- To develop architectural monitoring in heterogeneous, distributed, mobile systems
- To investigate relevant parts of autonomic computing in an AmI embedded context
- To include capabilities for discovery and control in the ambient intelligence world
- To incorporate self-healing, self-configuration, self-adaptation, etc. properties
- To enable self-adaptation of available information sources to the current context
- To explore the limits of centralised and decentralised architectures in this context

3.1.3 Hybrid architectures

Centralised architectures are not sufficient to fully support AmI embedded systems, neither are fully distributed architectures. Rather, hybrid architectures and adaptation to different types of architectures should be supported. The HYDRA middleware will thus be operational on both centralised and decentralised networks. The HYDRA research and development challenges and objectives are:

- To create and support a peer-to-peer/ad hoc architecture
- To operate on both centralised and decentralised networks
- To support adaptation of services and systems to different architecture models

3.1.4 Wireless devices & networks

Middleware for wireless objects should be able to hide the complexity of the underlying infrastructure while providing open interfaces to third parties for application development and ease of use for end-users. In HYDRA, the communication layer is not part of the middleware, which is transparent to it. The HYDRA research challenges and objectives are:

- To provide dynamic resource discovery and management
- To provide tools for advanced control making the solutions reactive to the physical world
- To incorporate semantics, which would allow object definition and querying for data
- To incorporate resources without any need for unique identifiers
- To provide unique network adapters in order to avoid specific networking technology

The HYDRA middleware will explore available network technologies in changing user-environments and discover available, location based information resources.

3.1.5 Trust and security

In order to solve the rapidly growing challenges of privacy, identity theft and trust, the HYDRA research objectives aim at two approaches in tandem:

The first approach targets the secure design and prototyping of mobile ID management for context-aware services relying on heterogeneous mobile and wireless service and networks. Security goals such as confidentiality, authenticity, and non-repudiation will be addressed by the particularly trustworthy design and implementation of open-source and web service-based mechanisms, enriched by ontologies and semantic resolution techniques.

The second approach reflects a coherent no-trust context-locked separation through virtualisation of devices and people[1].

HYDRA will thus refine a security and trust model. Multiple different virtualisation models with different security models will be assumed and mapping of interfaces will be carried out and considered in the overall architectural model. The model will specifically incorporate RFID with built-in access control and context-specific keys, virtual person ID and communication channel virtualisation.

3.2 Hydra Objectives in Establishing Benefit Realisation Models for the Hydra Platform

HYDRA research and development challenges and objectives are:

- Secure design and prototyping of mobile ID management for context-aware services, based on semantic identities.
- Design of for confidentiality, authenticity, and non-repudiation
- Implement virtualisation of devices and people.

In addition, an integral objective of the HYDRA project is the analysis and development of realistic business models for developers and service providers. New research into defining and measuring value creation in dynamic constellations will be undertaken, leading to innovative business structures involving content providers and service providers in collaborative systems.

Specific emphasis will be made on identifying new business opportunities for small and medium enterprises (SMEs). In particular SMEs with few products and limited resources find it difficult and expensive to embed ubiquitous intelligence in their products, because they need to communicate via, for example a GSM and the Internet. HYDRA will benefit SMEs because it will provide an open, secure, affordable and accessible framework for communication to their products and delivery of new, innovative services, including easy and open interaction with other manufacturers' products.

HYDRA will adopt an ontological perspective on the exploration of innovative service concepts and for quantifying value creation in complex products and value nets. The chosen approach is called e^{3}value[2], and is on the one hand based on the analysis of economic value creation, distribution, and consumption in a multi-actor network and on the other hand on requirements engineering and underlying conceptual modelling techniques. Thus, in this respect the HYDRA research and development challenges and objectives include:

- To define and identify value creation
- To model dynamic value constellations
- To identify sustainable business models in selected user domains
- To investigate the possible use of the HYDRA middleware for SMEs.

3.2.1 Application Domains

The Hydra middleware addresses two different types of users:

- Developer users, who will use the Hydra middleware
- End users, who will use Hydra applications developed by developer users

[1] Jacques Bus "Trust in the Net" http://wwwes.cs.utwente.nl/safe-nl/meetings/24-6-2005/jacques.pdf
[2] Jaap Gordijn: Value-based Requirements Engineering – exploring innovative e-commerce ideas, SIKS Dissertation Series no. 2002-8.

In order to properly evaluate user requirements for each of the three application domains, future scenarios were created using the IDON method. Scenarios provide coherent, comprehensive, internally consistent descriptions of plausible futures built on the imagined interaction of key trends.

The user domains provide the basis for realisation and validation of user applications in different real-world situations. At the same time they ensure a heterogeneous basis for requirement analysis as well as convincing showcases of the portability of the middleware across a wide range of domains.

The application domains were also used as a magnifier of specific functions and concepts of the Hydra middleware and were used as demonstration, dissemination and exploitation platforms.

3.2.1.1 Building Automation

Smart homes and Building Automation are buzzwords which cover a wide range of products, ideas and technologies. The application independent developed Hydra middleware fits perfectly in a wide range of domains including the home automation area. A variety of devices which operate in such environments were tested throughout the project. The main focused domain in the project was the smart home domain also using the latest technologies and devices including different network and communication standards. The new sub domain of energy efficiency was mostly covered with example applications in the scope of the Hydra project.

3.2.1.2 Healthcare

The Hydra middleware can be used in healthcare applications e.g. for enabling horizontal collaboration and client-centred responsibility in health risk management, self-management, homecare, and assisted living. National health-grids enable ambient intelligent services based on distributed computer- and human resources. Healthcare is thus a complicated but highly relevant area for the development and validation of Hydra applications.

3.2.1.3 Agriculture

As mentioned before the generic Hydra middleware can be used in any context where there are network embedded devices which need to interoperate with each other. During the project different scenarios in the agriculture domain were examined and tested.

3.3 Hydra Architecture

The software architecture described is an abstract representation of the software part of the Hydra middleware. The architecture is a partitioning scheme, describing components and their interaction with each other. Figure 1 gives a structural overview of the Hydra middleware and explains how the elements are logically grouped together. “Hydra managers” constitute the major building blocks that make up the Hydra middleware. A Hydra manager encapsulates a set of operations and data that realise a specific functionality.

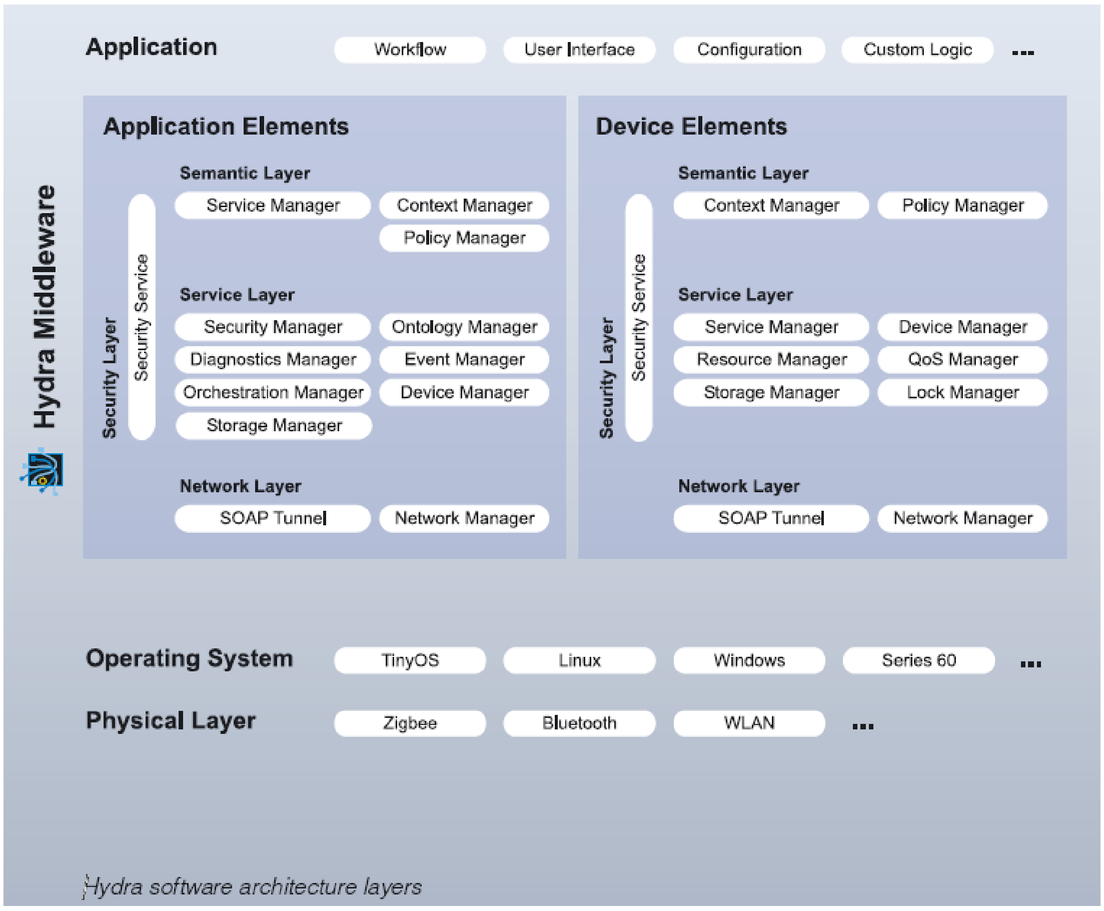


Figure 1: Hydra Architecture (layer model)

The Hydra middleware managers are enclosed by the physical communication layer and the application layer shown at the bottom and at the top of the diagram respectively. The physical layer incorporates several network connection technologies such as ZigBee, Bluetooth or WLAN. The application layer contains user applications which could comprise modules like workflow management, user interface, custom logic and configuration details. These two layers are not part of the Hydra middleware.

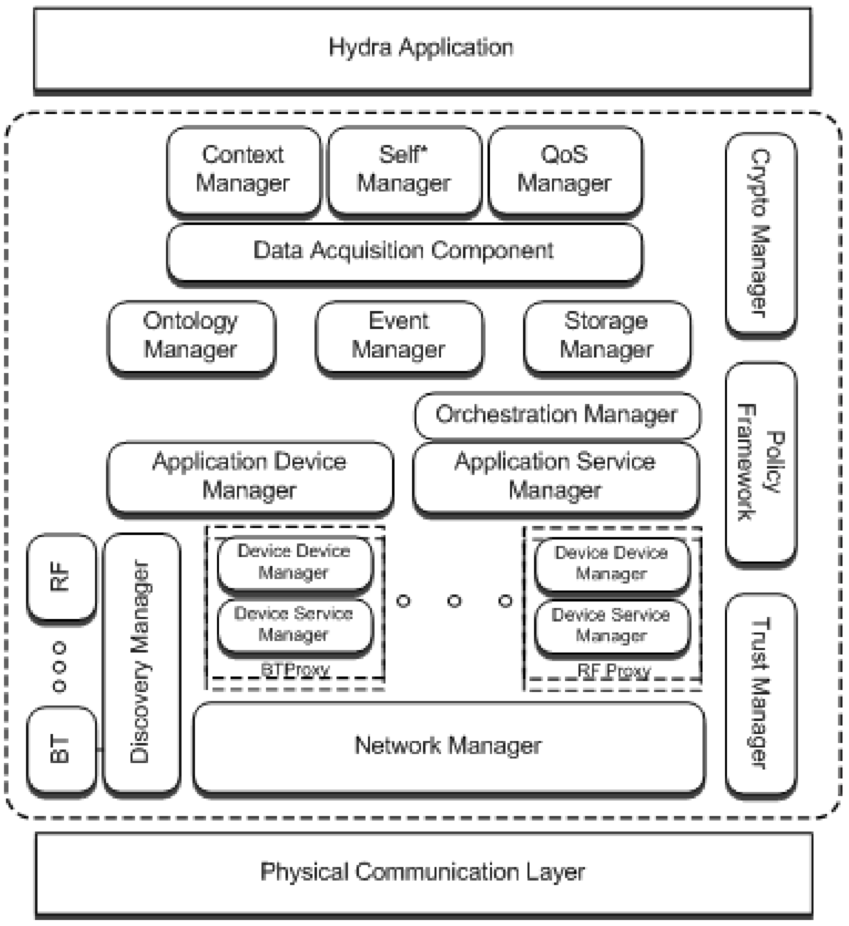


Figure 2: Structural Overview on the Hydra Managers

The Hydra middleware offers a large collection of reusable core software components to experienced developers. Based on these software components, programming abstractions allow for programming with well-known concepts from the field of pervasive and ambient computing through reducing the details of the underlying implementation. From the bottom to the top of Figure 2 the Hydra middleware provides more and more programming abstraction and functionality for the developers:

- The Network Manager implements Web Service over JXTA as the Peer-to-Peer model for device-to-device communication.
- The Device and Device Service Manager in a bundle implement a service interface for a physical device, handle several service requests and manage the responses.
- The Application Device and Application Service Manager provide programming interfaces and information for the different devices to the software developers.
- The Discovery Manager automates and facilitates the discovery of devices in a Hydra network.
- The Ontology Manager is used by the Application Device Manager to get meta-information about devices and also semantically resolves what type of device has been discovered.
- The Event Manager provides a topic based publish-subscribe service in Hydra.
- The Crypto, Trust and Policy Manager take care for cryptographic operations, the evaluation of trust in different tokens and the enforcement of access control security policies.
- The Data Acquisition Component retrieves the data delivered by the sensors (via push or pull mode).
- The Quality-of-Service (QoS) Manager in Hydra is a component that accesses and particularly processes all non-functional properties-data for services/components, devices, and network.
- The Self* Manager provides support for automating application management.
- The Context Manager allows for the definition of an application-dependent context model.
- The Storage Architecture realises the persistent storage of information in Hydra middleware.

3.3.1 Device Classification

The Hydra middleware is designed to handle all types of devices, with varying capabilities. The figure below, demonstrates how devices are classified into different categories, based on what technologies they can support, which determines how the device can become "Hydra-enabled".

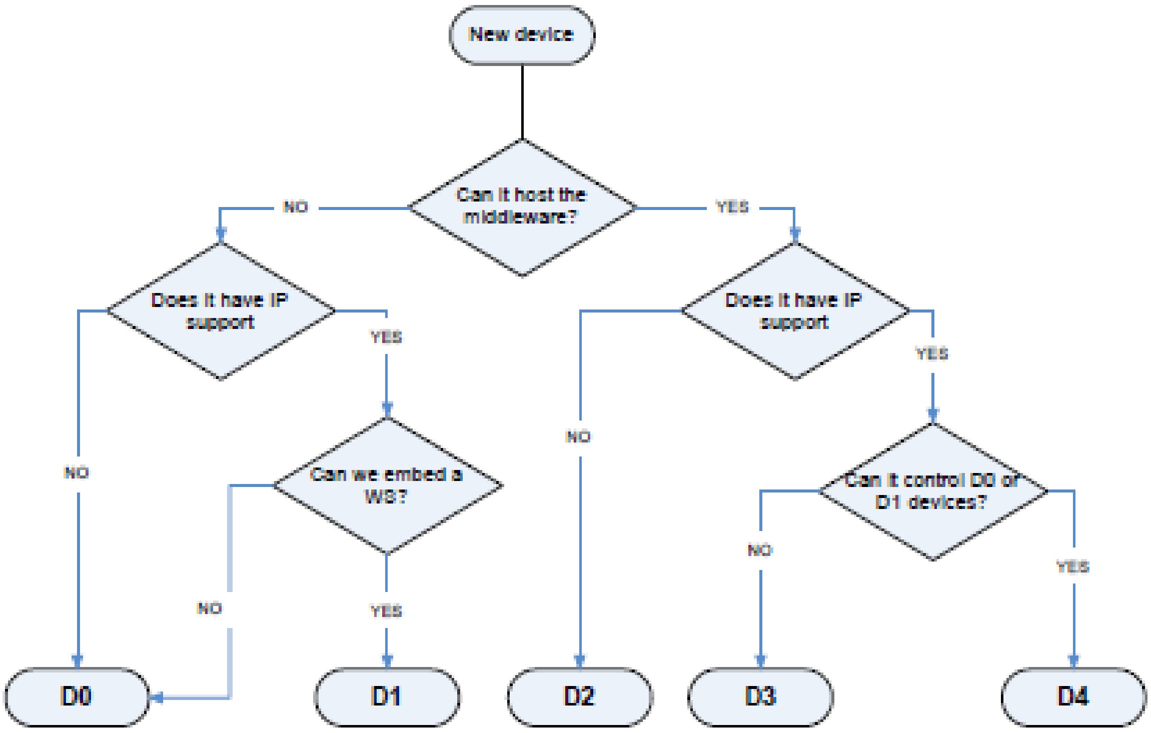


Figure 3: Flowchart for the device classification process.

The significance of the D0--D4 categories is that devices within each category are handled in the same way in relation to the Hydra middleware and the enabling process. For further details on Hydra terminologies please refer to the Glossary section in Chapter 8.

Category-D0 devices are used with a proxy, that is, they can only be reached through a proxy-service residing on a Category-D4 device. The proxy service must implement the communication with the D0 device.

Category-D1 devices can host a web service, and the intention is that such embedded web services are created with the Limbo tool.

Category-D2--D4 devices are said to be Hydra enabled. Hydra enabled devices host the network manager and all other managers needed for that device, but differ in their networking capabilities.

4. Hydra Technologies

This section summarises the technologies that Hydra utilises in order to realise its goals.

4.1 Java

Java is the core programming language of the Hydra Middleware, as it offers "write once, run anywhere" benefits. Using Java, the Network Manager can run on any Java enabled operative system. The Java J2SE can be run on almost any OS; therefore, Hydra can be deployed on these OS also. The problem of Java is the resources (memory and processor) that it consumes. It is not a big problem in powerful devices such as PC or Laptops, but in smaller devices, such as PDAs or mobile phones it makes it almost impossible to run the Hydra middleware on them.

4.1.1 OSGi

Most of the core components in the Hydra Middleware are implemented as OSGi plug-ins [11] because it offers a general-purpose, secure and managed Java framework that supports the deployment of extensible applications (bundles), bundle life cycle management, service oriented architecture and easy and quick deployment. Due to the use of OSGi in Hydra, deployment of all managers and proxies is simple and quick and its life cycle can be easily managed, even remotely. OSGIs modularity offers the developers a clean workspace where the managers can be dynamically deployed and un-deployed. Also, the service-oriented foundations are suited for the software architecture that Hydra was aiming for. Hydra also uses Remote-OSGi (R-OSGI) [12] which is an additional library that allows remote access to OSGi bundles. It uses less effort than using Web Services when component is based on OSGi and "full access" is intended. It is used mainly to connect the Hydra IDE to remote bundles for configuration and provides easy remote access to Hydra components.

4.1.2 .NET

Although most components of the middleware are implemented in Java, some are also implemented using the .NET framework. .NET provides good XML support and good development tools including cross process debugging. It has an Extendable XSLT processor and fast managers with a small memory foot print. Furthermore, LINQ [17] is a set of extensions to the .NET Framework that extends C# with native language syntax for queries and provides class libraries to take advantage of these capabilities [13]. In order to solve the platform support a port to Mono could be considered. Mono is an open source project aimed at bringing .net to as many platforms as possible. None of the .Net technologies used by the managers contains any functionality not available in Mono [14].

4.2 Network

Hydra is based on various network technologies and concepts which are briefly described below.

4.2.1 P2P / JXTA

A P2P architecture was selected for carrying out the communications between Hydra components (Inside Hydra Communications). The main benefits of a P2P architecture versus an old-fashioned client-server architecture are its adaptability to very extensible networks, the responsibilities are distributed among peers; it provides high availability and fault tolerance and enables the full usages of the bandwidth.

Among all the available P2P technologies, JXTA was selected as the most suited for Hydra. The reasons that led to the selection of JXTA are:

- Interoperability: Enables communication between peers independently of network addressing and physical protocols.

- Platform independence: JXTA does not depend on the programming language, network transport protocols and deployment platforms, giving freedom of choice. Java SE and Java ME implementations have been selected for Hydra.
- Ubiquity: JXTA is designed to be deployed on any device, not just PCs.
- Security: for security means regarding authentication, authorisation, and integrity can be implemented based on JXTA. Attacks on the level of the protocol cannot be addressed, as that would require changing the JXTA protocol.
- Community support: JXTA is supported by a wide community of developers and the different specifications are fully documented.
- Wide range of services: Most of the P2P models studied have been designed exclusively for providing file-sharing services. Instead, in JXTA, thanks to its abstract architecture based on six protocols, it is possible and feasible to create a wide range of interoperable services and applications.

Using P2P, Hydra solves the problems of traditional client-server architectures, providing communication even when the different middleware instances are deployed behind firewalls and NATs. Therefore, managers, devices and services can intercommunicate and interoperate.

Also, WS has been combined with P2P communications through SOAP Tunnelling in order to enable access to services and resources in a transparent way for developers. Using this SOAP tunnel, applications and devices can interoperate transparently, even when they are located in different networks, isolated from each other.

4.2.2 SOAP

Axis 1.4 for SOAP and Web Services provides a well working WSDL parser and generator and it can be easily transformed into an OSGi bundle. Also we selected it because it is compatible with the Limbo tool (see chapter 6.3; Axis 2.0 is not fully compatible). Axis 1.4 provides interoperability for service provision and consumption. However, the main problem with Axis 1.4 is the resources it consumes.

4.3 Semantics

Hydra uses current technologies and concepts to represent semantics in various parts of the middleware.

4.3.1 Context Awareness

The Context Awareness Framework, and namely its Context Manager component, uses a Rule Engine to provide the pluggable logic for reasoning and interpretation of data from devices. The Rule Engine used is the Java-based Drools Rule Engine [4], which is a powerful open-source Business Rules Management System, providing various functionalities that can be utilised, as described in the section of this document regarding the Context Awareness components.

4.3.2 Ontology

When more systems share the same language, they use the same terms and relations between the terms to describe the same topics. In this case, it is quite straightforward to exchange the information about required content between those systems. In knowledge modelling, the ontology represents the language used by systems. The more ontologies are re-used, the simpler the communication between the systems. The way of sharing the same language is the direct sharing of ontologies or reusing the same third-party ontologies. Re-using the third party ontologies generally tends to increase the semantic interoperability between the systems.

Third-party ontologies usually describe the specific part of information and have to be connected to the core ontology model. This is done by creating the relations between the relevant concepts.

As a formal language for creating the ontologies, OWL-Lite (and in some cases OWL-DL) was used. OWL-Lite and OWL-DL are a subset of OWL-Full language. The main arguments for selection of the formal modelling languages are the expression power and the real-time performance (taking into account the complexity of possible inferences in the model). RDF/RDFS are too simple, OWL-Full is too complex. OWL-Lite/OWL-DL is known to be a working compromise for a modelling language with satisfactory expression power and real-time performance.

The static ontologies were extended with SWRL rules. SPARQL and SQWRL were used as query languages. As the services provided by devices can be semantically annotated, SAWSDL standard was used as the semantic WSDL extension. SWRL rule language is the standard which is declarative, general, powerful, and can be natively integrated with OWL ontologies. Following the requirements on Hydra models, SWRL fitted the most of other possible rule languages. SPARQL and SQWRL query languages are the most common query languages used in ontology modelling, and they are supported in most implementations for ontologies handling. Use of the SAWSDL standard was driven directly by the requirement for semantic annotation of Hydra services.

The OWL language of Semantic Web technology, as an alternative to conventional relational database approaches, brings the flexible way of domain modelling in several points. It is based upon the open world assumption (OWA), which enables the flexible extensions and integration of domain models, many times in unexpected cases, which could not be planned in the phase of model design. Uniform representation language enables easy integration of the new knowledge, so the model can be extended without extra effort.

Uniformity of representation is also an advantage when using the query languages, such as SPARQL or SQWRL. This class of languages are based on matching the graph patterns, so in many cases, the extension of knowledge does not have to affect the query already used, the query automatically returns the added knowledge (this is the OWA, but also the good model design advantage). Another advantage of Semantic Web modelling languages is the use of inferred knowledge, which means that, the languages have a strong expression power enabling them to deduce the facts, which are not explicitly contained in the model. Ontologies in Hydra are composed of several Hydra specific models, but also of external integrated third party ontologies. The inference and graph patterns matching is used in several Hydra specific tasks, such as resolution of semantically annotated services, an QoS properties based search or semantic device discovery.

Of course, the uniformity of knowledge models representation is also the big advantage in the process of domain models standardisation (reusing existing knowledge) and the knowledge exchange, which leads to the semantic interoperability between the components of applications, or the applications themselves.

SWRL language is used to formulate the rules above the Hydra ontologies mainly in tasks of self-* diagnosis or resolution of the security properties of Hydra devices. As there is a need to formulate causal logic relations and implications, the rule language is required for certain tasks. The possibility of using the rule language, which can be natively integrated with the Hydra ontologies is an advantage in itself.

4.4 Security technologies

A description of security technologies used in Hydra are listed and described below.

4.4.1 XACML

XACML [8] is an OASIS [6] standard that establishes an XML-represented language for access control policies, as well as access requests and responses. It defines a processing model, as discussed in the introduction to this section. It has standard extension points for defining new functionalities, making it rather flexible and extensible, and as such is perfect for integration into the Hydra middleware.

4.4.2 XML Security and Web Service Security

The Web Services Security specification (WS-Security) [9] enables Web Services developers to secure SOAP message exchanges by providing them a set of mechanisms. WS-Security enhances existing SOAP messaging which provides quality of protection. This is done by applying integrity and confidentiality to messages and authentication to SOAP messages. Furthermore, WS-Security also provides a general mechanism which helps in associating security tokens with messages. The good thing is that WS-Security does not require a specific type of security token. WSS is extensible and supports a variety of authentication and authorisation mechanisms.

5. Hydra Components

This section summarises all Hydra components by providing an overview regarding the functionality and purpose of each component. Furthermore, the implementation of each component is described in detail focussing on the technologies used and the reasons for choosing these technologies. The value each component adds to the overall Hydra middleware is also discussed.

5.1 Network Manager

The Network Manager is responsible for network management. It is in charge of providing a transparent view of the nodes in the application and to route the data to the appropriate node (high-level view shown in Figure 4). Network communications are traced with a session mechanism. Furthermore, it takes advantage of the peer-to-peer architecture to create a Hydra overlay network and to allow the discovery of other Hydra-enabled devices. It also handles the HIDs (Hydra IDs) of the nodes in the application. Finally, it is in charge of synchronising the nodes in the network with referential time.

The Network Manager is the bottom layer of the Hydra middleware deployed in gateways and in Hydra-enabled devices. It is the entry and exit point of information of the Hydra middleware. There is only one Network Manager per device where the middleware is deployed. The Network Manager provides a web service interface which is the information entry point for the middleware. Data transferred between Hydra-enabled devices and gateways should always pass through the Network Manager.

Therefore, the main functions of the Network Manager are to provide a unique entry point for the network communications, support session mechanisms, create a peer-to-peer based overlay network, provide HID to nodes at application level, handle a list of the network members, allow discovery of other Network Managers and synchronise network with referential time.

The Network Manager is based on P2P and SOAP tunnelling combined with Web Services. The network manager builds an overlay P2P network as can be seen in Figure 5:

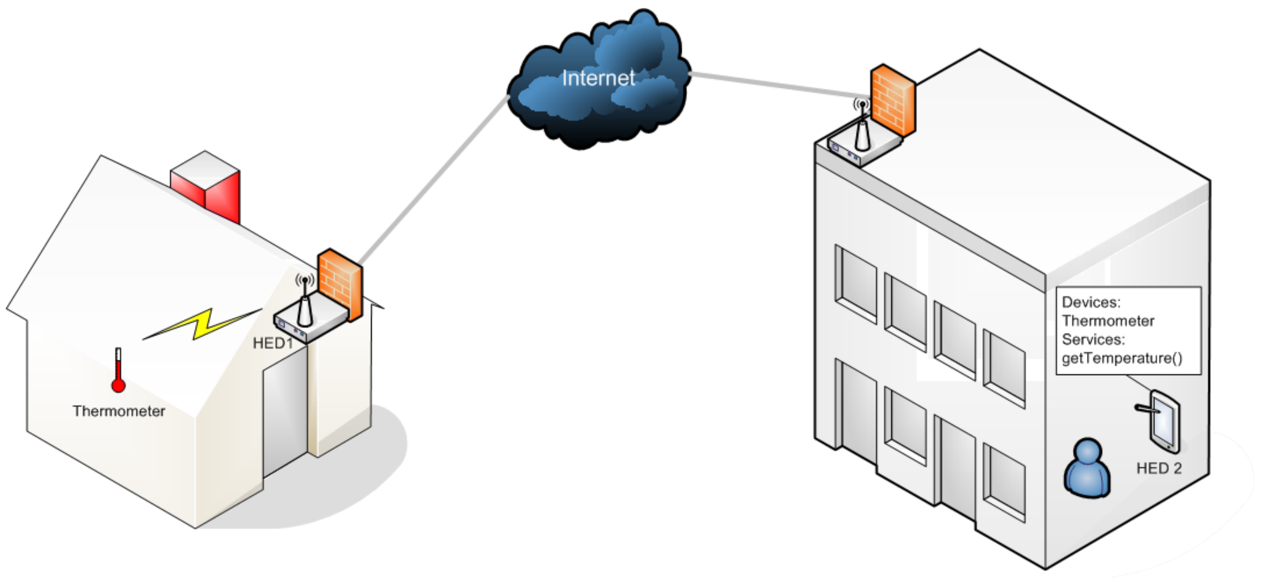


Figure 4: Network manager overlay P2P network

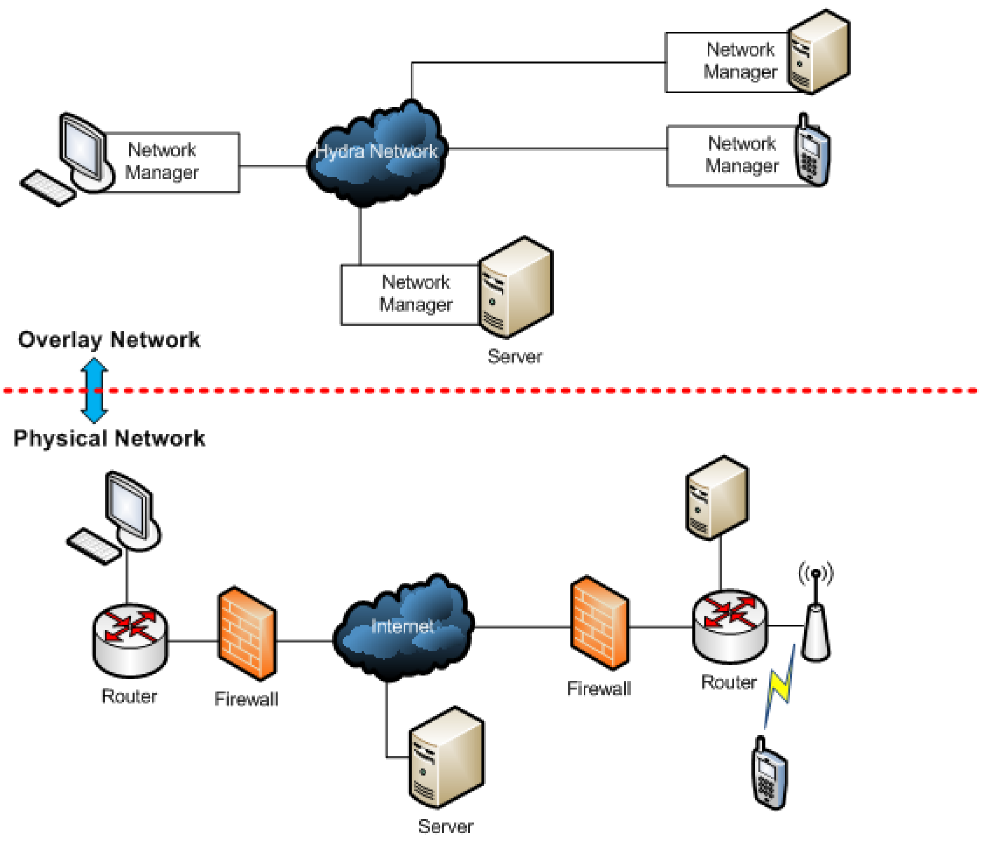


Figure 5: Basic Hydra network compared with physical network

5.1.1 HIDs and their management in Network Manager

The network manager allows services to be registered and made available through the Hydra Network with the use of Hydra IDs (HIDs). Each service from a device is registered in the local Network Manager and a unique context-dependant Hydra identifier (HID) is assigned to it. This does not depend on routing or network infrastructure, it can be kept even if the endpoint changes (mobility). The Network Manager provides the mechanisms for creating, modifying and deleting HIDs and ensures uniqueness. In order to register a service, the device (or proxy) provides a local endpoint of the service, an optional short description, and an optional Context or super context. The Network Manager maintains a data structure called IDTable with HID Info of services registered on it as well as HID Info (not the endpoint) of remote services registered on other Network Managers in the network.

5.1.2 Other network manager functionalities

A PEP (Policy Enforcement Point) is integrated in the Network Manager in order to enforce policies associated with HIDs (or crypto HIDs). The enforcement is performed before the service invocation. SOAP Tunnelling is carried out over BT and UDP by implementation of new Network Manager transport mechanisms (BT and UDP) for SOAP message delivery for last-mile communication (gateway<->device). Protocol switching is made possible by changing the endpoint for a service (HID) in order to switch between communication technologies -> TCP / UDP / BT. The Network Manager is an OSGi R4 compliant bundle and a Network Manager Service is exposed both through WS and OSGi service. Furthermore, the Network Manager is UPnP AV compliant and incorporates Inside Hydra Security, multimedia content exchange, and a separate communication mechanism based on P2P for multimedia content.

5.2 Event Manager

The Hydra Event Manager provides publish/subscribe functionality, i.e., the ability for publishers to send a notification to multiple subscribers while being decoupled from them (in terms of, for example, not holding direct references to subscribers). In general, publish/subscribe communication provides an application-level selected multicast that decouples senders and receivers in time, space, and data (i.e., sender and receivers do not need to up at the same time, do not need to know each other's network addresses and do not need to use the same data schema for events they send). The specific variant of publish/subscribe implemented is topic-based publish/subscribe where key/value pairs represent events. With this approach, any subscriber or publisher defines a topic simply by executing the "publish" or "subscribe" actions.

The Event Manager is used in any place where there is a potential many-to-many relationship between senders and receivers and where asynchronous communication is desirable. In particular, the Application Event Manager provides subscription support by allowing clients to subscribe to published events via a topic-based publish/subscribe scheme, and publication support by allowing clients to publish events on topics. The Event manager routes events to subscribed clients and assists in interfacing to the Network Manager (e.g., broadcast-, multicast-, or gossiping-based dissemination). The Event Core manages persistent subscriptions and publication to subscription matching etc. The Event Manager makes it possible to send and receive events through the Hydra P2P network.

Both the Event Manager and Resource Manager are programmed from scratch within Hydra, without the use of third party components beyond the standard Java libraries. All components make use of OSGi. The use of Java technology for all components was chosen because of the experience available in this language. Additionally, there is a wide selection of open source third party libraries available in Java. OSGi was chosen because of the modularity possibilities it affords.

5.3 Ontology Manager

Hydra ontology stores all information and knowledge regarding devices and device types. The ontology models are focused on description of the devices functionality and properties. The Hydra ontologies cover taxonomy of devices (Figure 6), device services (including service taxonomy), device discovery supporting models, Quality of Service features, Self-* properties supporting models, security model, HW and SW device properties, and device malfunctions.

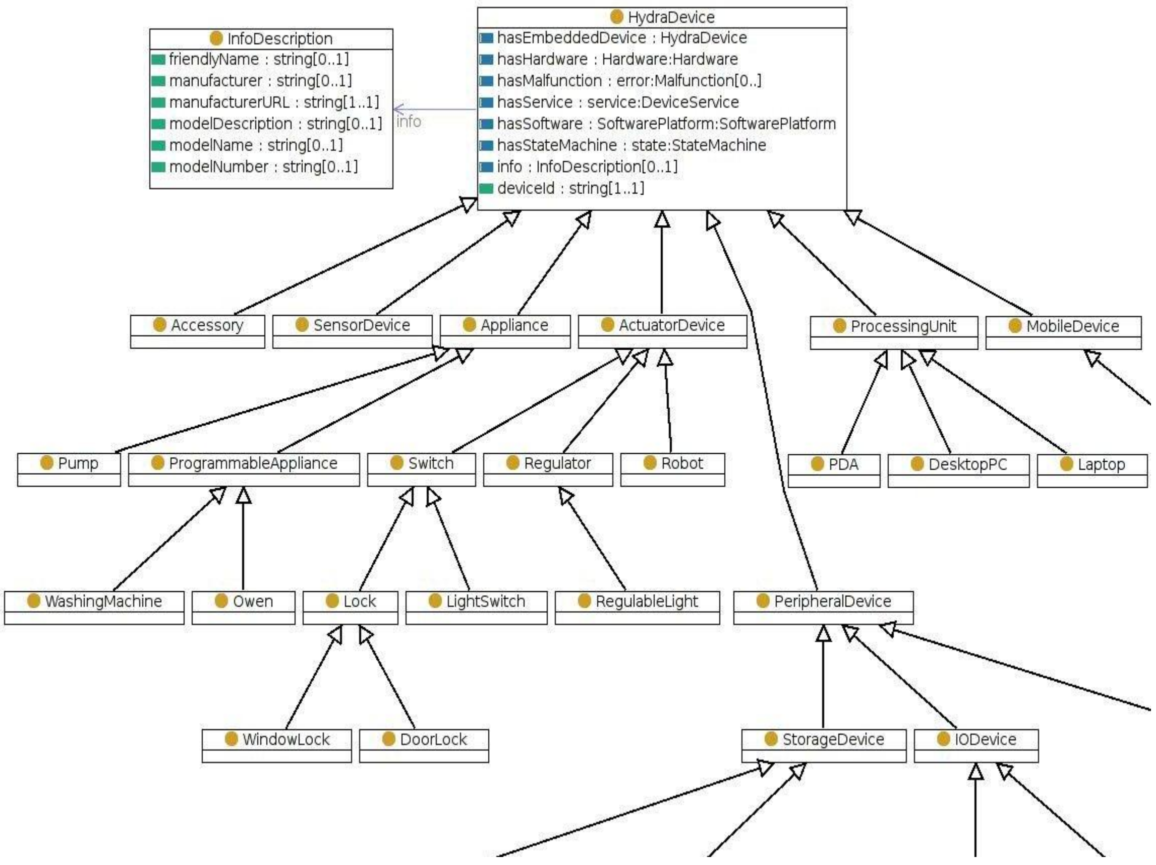

InfoDescription
friendlyName : string[0..1]
manufacturer : string[0..1]
manufacturerURL : string[1..1]
modelDescription : string[0..1]
modelName : string[0..1]
modelNumber : string[0..1]
info
HydraDevice
hasEmbeddedDevice : HydraDevice
hasHardware : Hardware:Hardware
hasMalfunction : error:Malfunction[0..]
hasService : service:DeviceService
hasSoftware : SoftwarePlatform:SoftwarePlatform
hasStateMachine : state:StateMachine
info : InfoDescription[0..1]
deviceId : string[1..1]
Accessory
SensorDevice
Appliance
ActuatorDevice
ProcessingUnit
MobileDevice
Pump
ProgrammableAppliance
Switch
Regulator
Robot
PDA
DesktopPC
Laptop
WashingMachine
Owen
Lock
LightSwitch
RegulableLight
PeripheralDevice
WindowLock
DoorLock
StorageDevice
IODevice

Figure 6: Hydra Device Ontology

The purpose of the application Ontology Manager is to provide an interface for using this Device Ontology. The Device Ontology uses a third-party ontology for description of security properties of devices and services. For this purpose, the NRL ontology [15] was used. NRL ontology contains the basic *securityConcept*, which was connected to the *HydraDevice* and *Service* concepts using a *hasSecurityProperty* relation. Generally, any third-party ontology may be used for specific representational purposes. It is enough to create the relation between ontology concepts. Once these relationships are made, ontology may be populated and the newly added concepts and relations may be used for specific semantic inference and searches. The Device Ontology also contains additional models addressing specific features of devices such as device hard- and software capabilities, device services, device state-machines, device security capabilities, and Quality-of-Service properties.

This manager also maintains the runtime instances of Hydra devices. The Ontology manager serves as the access point to HYDRA ontologies, provides general methods for ontologies maintenance and use by other Hydra components. It is implemented using the Protege API (Jena based) [16] with a Pellet running an in-memory model to enable fast real-time query performance and reasoning. In case of using large amounts of data, the underlying model storage can be changed to a custom database back-end or native triple store.

The main functionalities of the Ontology Manager can be summarised as:

- Describing and Annotating Devices: The core of the ontology handled by the Application Ontology Manager contains descriptions of devices. The knowledge model of these devices can be updated and refined. The new device models can be created as instances of this knowledge model.
- Parsing and Annotation of Device Description: The device descriptions can be updated using the third-party data (such as UPnP profiles). The data is parsed and the respective part of ontology is updated.
- Parsing and Annotation of Device Service Descriptions: The service models can be automatically generated or updated using the SAWSDL and WSDL W3C standards. The Application Ontology Manager parses the (SA)WSDL document and creates the related service models. These models can be updated following the structure of document and/or contained semantic annotations.
- Updating the Runtime Ontology: The ontology instances representing run-time device models can be continuously updated. The Application Ontology Manager provides the methods enabling the change of run-time ontology data.
- Searching and Querying: One of the basic features of knowledge models comprises the ability to answer the queries. The Application Ontology Manager provides specific querying functionality hard-coded in specific Web-Service methods. The queries are implemented in the SPARQL language. The Application Ontology Manager also implements a general query interface, which enables the developer to retrieve the result of any SPARQL query.
- Reasoning: The ontology queries require the inference in the provided knowledge model. A specific reasoner module supports the search and query functionality.

The State Machine ontology is used for the creation of state machine stubs to handle device run time status changes, and also for the diagnosis and monitoring rules used in order to achieve self* properties. The State Machine ontology models the state machine concept in UML2. The concepts in State Machine ontology are shown in the following Figure 7:

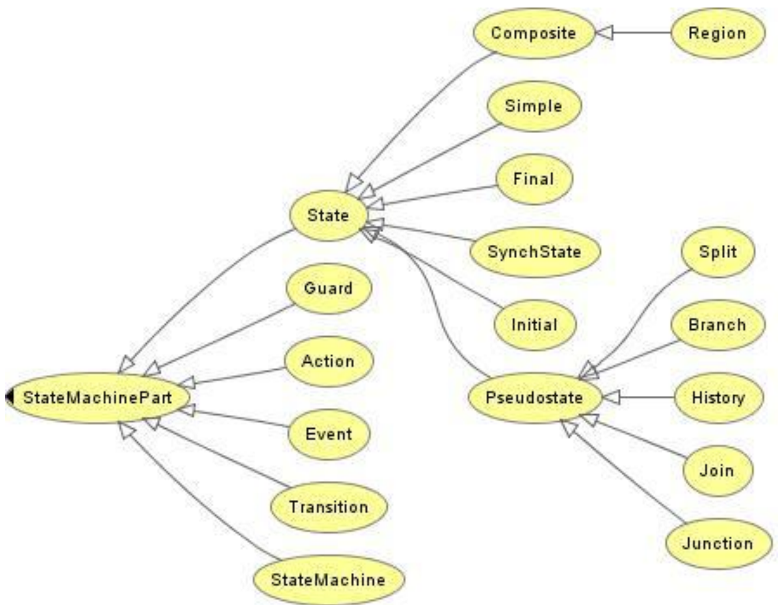


Figure 7: State Machine ontology

5.4 Trust Manager

The Trust Manager can be used to verify whether a token offered by an entity is trustworthy. The term “trust” is used in a very technical sense: "trust" indicates that the likelihood of an entity being the legitimate owner of a key inside a token. The methodology by which this likelihood is determined is called a "trust model". A trust model is an algorithm that takes a token as input and returns a trust value as output. Although this interface appears to be very simple, the process behind can be arbitrarily complex. The most common trust models such as Public Key Infrastructure (PKI) or Web of Trust (WoT) may require sophisticated checking of key chains.

Instances of the Trust Manager are used in several places inside the Hydra architecture. It is a useful tool for an application developer to verify application layer certificates, but it is also an important component for middleware layer security. Thus a device developer has to integrate the Trust Manager in a device in order to support device certificate management for the secure middleware communication protocols. Alternatively, device software can interact with the Trust Manager as an external certificate verification service if the computational power of the device is insufficient to run a Trust Manager. Therefore the Trust Manager offers its functionality as a web service. In this case, the communication between device and Trust Manager has to be protected using the Core Hydra security mechanism or by other device specific means. The Trust Manager acts as an interface to arbitrary trust models, for example Public Key Infrastructures (PKI), Web of Trusts (WoT) or reputation-based trust models. Which specific trust model should be used depends strongly on the application context the device will be used in and cannot be predetermined.

Thus the Trust Manager implements not a specific trust model but rather provides an interface so that different models can be used without requiring any change on the program code. A device developer can use one of the preconfigured trust models, which are currently X.509 PKI, OpenPGP (Web of Trust), or a null-model (trust every certificate, for development). Furthermore developers are free to implement their own trust models and are able to add their model only by changing a configuration file. During runtime, the trust models can be selected and configured via a web service if the device developer wants to open this functionality.

5.5 Crypto Manager

The Crypto Manager is a stand-alone manager providing various cryptographic operations such as encryption, key management and handling of digital signatures. The main functionalities of the Crypto Manager are to protect messages, encrypt, sign, decrypt, verify signatures, wrap data in different message formats, receive protected messages, manage keys, generate public/private key pairs with or without persistent identifiers contained, generate symmetric keys, and store public keys (certificates) as well as symmetric keys.

The Crypto Manager is basically used in two ways: On the one hand, the Crypto Manager is automatically used by an internal part of the Hydra middleware – such as the security modules in the Network Manager. Therefore it provides cryptographic operations as an internal part of the middleware. For example, in the case that a device is about to join a Hydra security domain, a secret shared key has to be agreed between the device and the initiator of the domain (the "domain controller"). This functionality is located in the Crypto Manager, so it has to be attached to the Network Manager (either via web service calls or, better, directly as an OSGi bundle). This way, it is very easy to exchange sensitive cryptographic operations at a later time without touching any other components.

On the other hand, it can be used by application developers via its web service interface as a stand-alone component for key management, and the creation of protected messages, thereby allowing building of secure storage solutions or communication protection on the application layer.

The Crypto Manager provides its methods as a Web Service as well as an OSGi service. It is strongly recommended to use the Crypto Manager over a connection that is not prone to eavesdropping and modification. That is, recommended ways to use the Crypto Manager are either locally as an OSGi service or remotely over a channel that has been protected by Core- or Inside Hydra mechanisms, for example.

5.6 Context Awareness Framework

The Context Awareness Framework provides a link between device-sensed data and semantics in the Hydra middleware, aiming to support developers with a means to utilise context-processing capabilities into a Hydra-enabled application. The Context Awareness Framework contains components to retrieve data from the Hydra environment, providing a base for enabling interpretation of context information along with context-sensitive, application-defined actions.

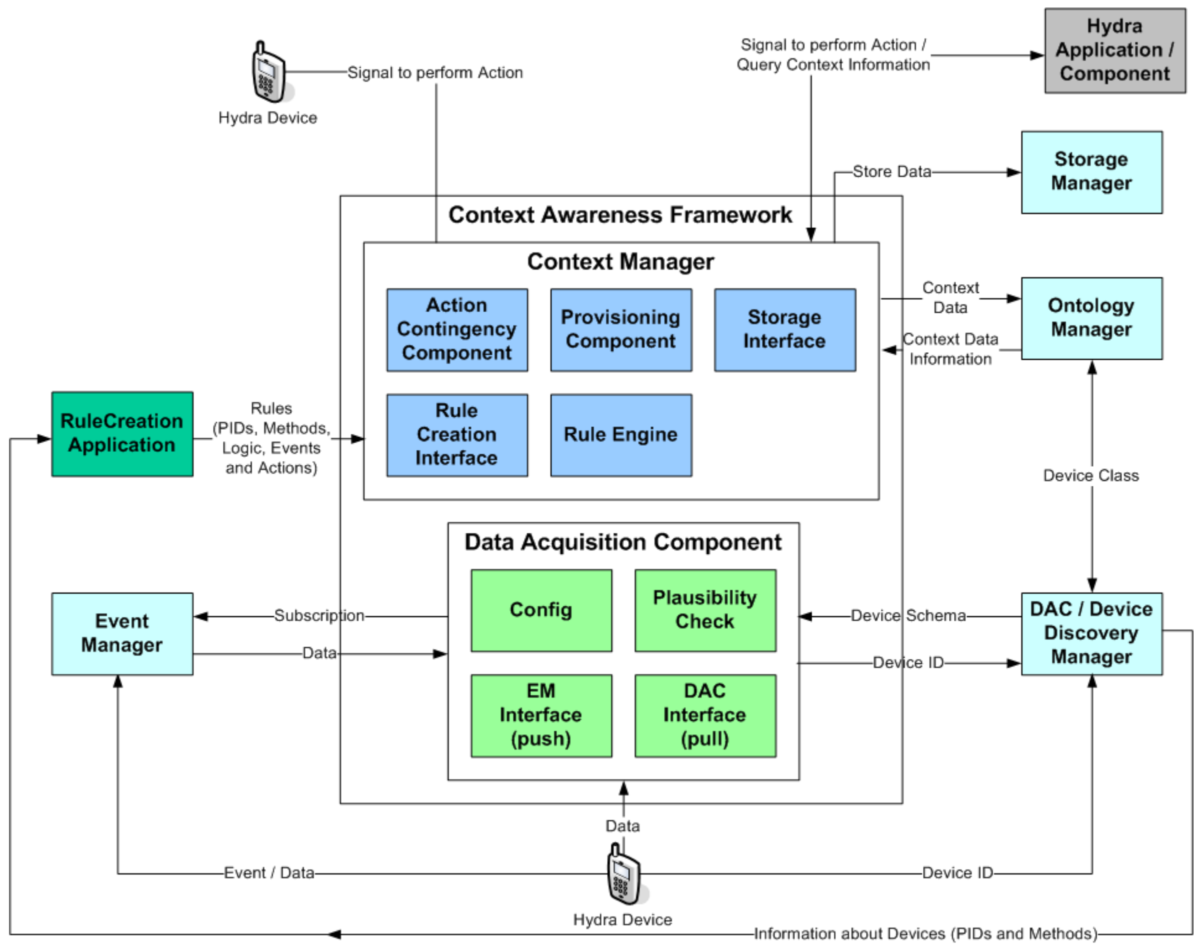


Figure 8: Context Awareness Framework and Hydra

Figure 8 shows the components of the Context Awareness Framework, and how they integrate with other components in the Hydra Middleware. As can be seen, there are two central components that provide the functionality of Context Awareness in Hydra.

5.6.1 Data Acquisition Component

The *Data Acquisition Component* (DAqC) is a lower-level component that is responsible for retrieving data as configured, from a variety of sources. It receives subscriptions for data from data sources, and fulfils them, using the protocol specified. Subscribers to data from the DAqC, must implement the *DaqcReportingService* interface, so that the DAqC can report the acquired data back. Data reported is packaged in a uniform manner, with a given data id that is specified at the time of the subscription by the subscribing entity. The protocol used to retrieve the data is passed, as well as the data itself.

Typically, there are two main protocols for retrieving data:

- *PUSH*
  - Handles events published to an Event Manager by a data source
- *PULL*
  - Handles the retrieval of data directly from a device

The *Push* protocol handles data that is provided in an asynchronous manner, typically by the data source publishing it to the Event Manager. Therefore, the *Push* protocol essentially acts as a wrapper to the Event Manager, for the entity subscribing to the DAqC, simply forwarding the event on to the DAqC subscriber, after packaging it appropriately for reporting as DAqC-acquired data.

The *Pull* protocol handles data that needs to be actively retrieved from a device - most typically sensor information. The DAqC sets up processes for retrieving this data at the specified frequency, the results of which are reported to the subscriber.

Additionally, the DAqC may perform some plausibility checking (if configured to) of the retrieved data, using Regular Expressions that are included in the subscription to the DAqC, to attempt to discover situations in which the data source may have gone rogue, and no longer be a viable source of contextual information. Any failures are reported in the Data Report sent to the subscriber when new data is acquired, and the subscriber has the responsibility of acting on it - potentially by cancelling the subscription altogether. Other failures are reported to subscribers, such as failures with the *Pull* protocol in retrieving data from a device, potentially because the device has gone offline, and is no longer available as a data source.

5.6.2 Context Manager

The *Context Manager* is the higher-level component of the Context Awareness Framework that is responsible for management of contexts, as well as providing a mechanism for interpretation of data into contextual data, and a mechanism for reasoning over contextual data and to perform context-sensitive actions. It stores "Current" contextual information, and can be configured to store historic contextual information that itself can be queried. The Context Manager is designed to handle the transformation of low-level *key-value* data, retrieved through the DAqC, into "marked-up" contextualised data, such that it is of value to a Context-aware application. It also serves as an *Event Processor* to handle and interpret "Events", to contextualise them, and to reason over them from both point-in-time, and temporal, point of view.

The Context Manager, as shown in Figure 8, has a Rule Engine as a core component that provides the functionalities for the interpretation of data into contextual data. This is the Drools Rule Engine [4], which is an open-source Business Rules Management System, which brings together several features that are used to model, interpret and reasoning over the contextual data. The "Working Memory" approach of the rule engine is utilised to model context data internally in an object-oriented manner, combined with the *key-value* approach, which equates to the notion of *Current Context*.

Additionally, Drools provides *Complex Event Processing* capabilities, through its Drools Fusion module, that enables rules in the Context Manager to reasoning over events using Sliding Windows and / or Temporal reasoning, enabling rules to contextualise states or data by using an extra dimension - time.

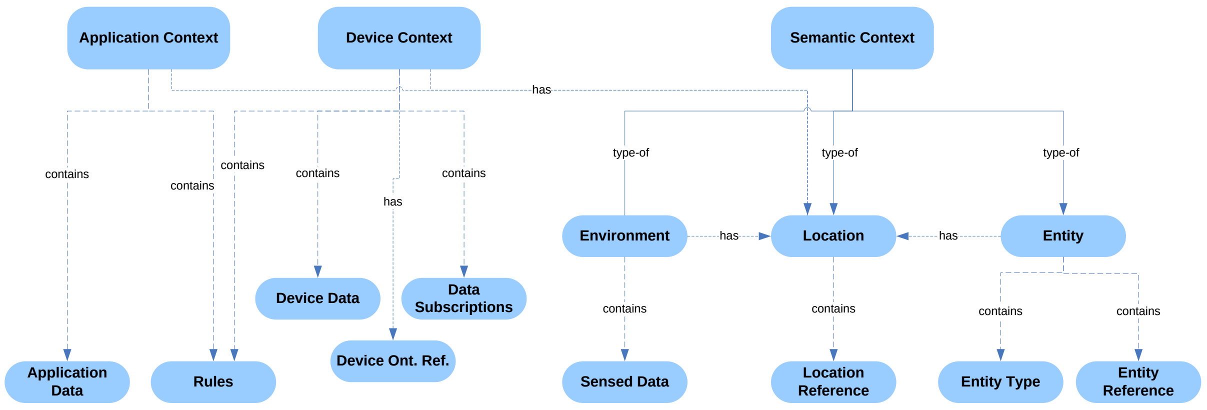


Figure 9: Structure of Contexts

As Figure 9 shows, three core types of context are handled by the Context Manager. These are:

- Semantic Contexts
- Application Context
- Device Context

Semantic Contexts are used to store instances of semantic properties in the Context Manager to be used for interpretation. Although this "ontology" has been defined as above, it is extensible to add new kinds of semantic concepts easily. The key notion for contextualising data from a data source, is knowing the *Location* of the data that has been sensed, so that it can be attributed to an appropriate context. The *Environment* context represents a collection of data that has been sensed in a particular environment. It has a defined *Location* property such that if a context-aware application requests a set of data about an environment, or even just a particular type of data about an environment (for example, humidity), then this is easily retrievable. Similarly, device data sources themselves maintain a *Location* property, that allows the data that they sense to be interpreted as being a property of the environment of the location that they are in.

Device Contexts represent the context of a device, including references to the Device Ontology providing a definition of the device, as well as storing the latest values retrieved from the device - if it is a provider of sensed data. They contain subscriptions for data that are sent to the DAqC, as well as a status property that maintains the current status of the device with regards to its existence and availability as a data source.

Application Contexts are sets of rules and collected data, that are used for a variety of purposes, including performing interpretation of data to build and maintain semantic concepts stored in the Context Manager, or for more application-level purposes. This could include collecting data for a specific purpose that may be stored or queried as a package, or using the rules to reason over situations, as a collection of data values, states and events, to make context-sensitive actions. For example, a simple context rule could specify that when the lighting in environment "A" is dark, to close the curtains in that room.

5.6.3 Context Provisioning

A key feature in using the Context Manager to create context-aware applications is the provisioning of context data. This can either be the storage of context data to the Storage Architecture, or by responding to a context query. One example of using this within the Hydra Middleware itself, is given in Figure 10, where contextual information can be used to create context-aware access control policies.

5.6.4 Context and Ontology

The Context Manager handles data in a fairly low level manner, and then adds value to it through rules that contextualise data to provide higher semantic information. The information represented must still be referred to utilising a "common" language, such as with Ontology. Device Contexts, as discussed in 0, contain a property that acts as a link to the Device Ontology, to associate the device with a higher level concept modelled in an Ontology. The same approach exists with all data that figures in the Semantic Contexts, where their data members have value added to them by defining what they represent with reference to a particular ontology. For example, temperature data represented in Celsius form may be retrieved from a device, and can then be contextualised as a property of the environment in which it was sensed - with the mark-up reference to define that it is Temperature->Celsius that is being represented by the associated data value.

The Context Manager and Ontology Manager may also co-operate to share information, so that higher level reasoning, using the more powerful descriptive and semantic techniques of OWL and SWRL etc, can be performed using the contextual information sensed at a lower level. Likewise, the Ontology Manager can inform the Context Manager of updates to static semantic instances, such as the change in location of a device that would then directly affect the semantic interpretation of the data retrieved from the device.

5.7 Access Control Policy Framework

The Access Control Policy Framework, in the Hydra Middleware, is an implementation of the XACML (eXtensible Access Control Mark-up Language) processing model [8]. It adds the functionality to be able to protect Hydra devices and services from unauthorised access, at the network level. The XACML Processing Model consists of four main components:

- Policy Enforcement Point (PEP)
    - Intercepts the call before it reaches the resource, and formulates an access request using the known information about the involved entities. Sends the request to the PDP. Enforces the decision returned, and may perform any obligations returned.
- Policy Decision Point (PDP)
    - Makes a decision on the access request, again the repository of policies it holds. The decision is returned to the PEP.
- Policy Information Point (PIP)
    - Resolves attributes referenced in a policy, that are not featured in the access request.
- Policy Administration Point (PAP)
    - Provides the point of administration for authoring, publishing and managing XACML policies on a PDP.

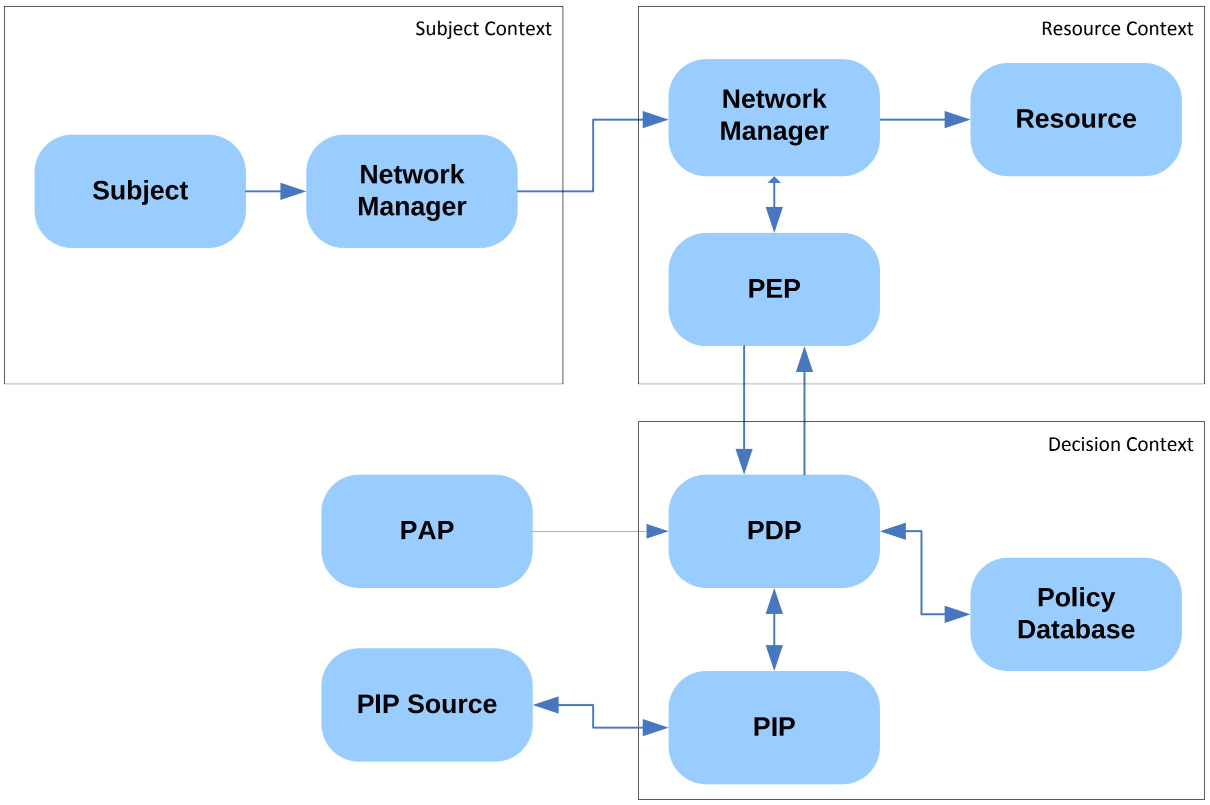


Figure 10: Access Control Policy Framework

Figure 10 above shows the architecture of the Access Control Policy Framework in Hydra. As described in 5.1, communication between *Subject* and *Resource* is routed through the Network Manager and Soap Tunnelling, with the Network Manager of the *Resource Context* being that which hosts the *Resource* service on the Hydra network, which can forward the request to the resource. Before doing so, it must request an access decision, passing all credentials of the request to the PEP. This includes:

- Subject and Resource HIDs
- Method being called
- Session ID
- CryptoHID attributes of Subject and Resource

The PEP formulates this information into an XACML Request Context document, which is then sent to the PDP for a decision to be made. It should be noted that the PDP may or may not be local to the PEP. In the case where the PDP is remote, communication between the PEP and the PDP is again routed through the Network Manager.

Upon receiving the request, the PDP attempts to retrieve any relevant policies from the Policy Repository, which is a local XMLDB. The request is evaluated, which may involve retrieving additional information using a PIP, and a response returned with a decision and possibly a set of *Obligations*. The possible decisions returned are:

- *Permit*
  - Access granted
- *Deny*
  - Access denied
- *Indeterminate*
  - A decision could not be made, potentially due to an error in evaluation
- *Not Applicable*
  - No relevant Policy was found for a decision to be made

When the PEP receives the response, it handles any obligations returned, and returns the decision to the Network Manager, as the PEP itself does not have the power to technically enforce the decision itself, but relies upon the Network Manager to perform this role.

The final component of the Access Control Policy Framework is the PAP, which provides the interface for authoring, publishing and managing XACML policies on a PDP. In Hydra, this is implemented with the *Access Control Policy IDE* - a component of the Hydra IDE.

The PDP uses an XML Database for storing policies published to it, as XACML policies are defined by XML anyway, and the ability to use XPath and XQuery queries to retrieve relevant policy(s) from the database make for a much more efficient system, as opposed to iterating through policies for matches. The database used is the eXist XMLDB database [5].

5.7.1 XACML

XACML is an OASIS [6] standard that establishes an XML-represented language for access control policies, as well as access requests and responses. It defines a processing model, as discussed in the introduction to this section. It has standard extension points for defining new functionalities, making it rather flexible and extensible, and as such is perfect for integration into the Hydra middleware.

The Access Control Policy Framework in Hydra is based upon the XACML implementation by Sun Microsystems [7], which is the commonly used XACML implementation in Java. It provides the core components of XACML, with all the core subset of XACML data types, functions and algorithms implemented, and is designed to be as easily extensible as the XACML standard itself. The Sun implementation is based on the XAML 1.x specification, with some components of XACML 2.0. XACML 3.0 and is still in the drafting phase.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy PolicyId="ExamplePolicy"
      RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:ordered-permit-overrides">
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">MyResource
               </AttributeValue>
          <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                  AttributeId="hydra:policy:resource:pid"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
```

```
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">doSomething
            </AttributeValue>
            <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Rule RuleId="OnlyAllowMySubject" Effect="Permit">
      <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                      AttributeId=" hydra:policy:subject:pid "/>
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">MySubject
             </AttributeValue>
      </Condition>
    </Rule>
    <Rule RuleId="CatchAllDeny" Effect="Deny"/>
  </Policy>
```

Listing 1: A simple XACML Policy

The listing above shows a simple XACML 1.x policy, that protects the "doSomethiing" method of a resource with a PID of "MyResource", against access from anything but a caller with a PID "MySubject".

As mentioned in the introduction to this section, the Access Control Policy Framework has its main point-of-entry to the Hydra Middleware at the Network Manager level, advising the Network Manager on what action to take when it receives a call to one of its hosted services. The other main point of interaction is through the use of PIPs.

The Hydra PDP is designed to be extensible, to easily allow for new functionality to the PDP through adding additional PIP components, which includes the ability to resolve certain attributes, add additional functions that can be used in policies, add new data types, and so on. For example, the *Context Manager PIP (Chapter* 5.7) provides functions for querying contextual data from a Context Manager, to be used in a policy, allowing for *Context-Aware* policies.

PIPs, as with all components in the Access Control Policy Framework, are implemented as OSGi bundles, implementing an interface exposed by the PDP such that it is very easy to add new PIPs.

5.8 Obligation Framework

The Obligation Framework in Hydra, is a realisation of the event-condition-action (ECA) policies, using some advanced techniques such as semantic reasoning, complex event processing (CEP) and enforcement monitors to increase the benefits of the policy framework. The purpose of these policies is manifold: Obligation policies shall help developers in setting up a Hydra-based system that automatically adapts its settings and implementations upon context changes (including, but not limited to security settings). Further, they shall complement the access-control policies by adding more expressiveness than simple permit/deny decisions. Last but not least, obligation policies can be defined by end-users (meaning: users of the Hydra application) in order to define the behaviour of their devices depending on different situations.

In XACML, it is possible to specify an obligation element for each policy and define whether the obligations contained within must be executed upon a deny or permit decision [9]. So, once the PDP has evaluated an access request, it will send the decision to the PEP, along with the set of obligations of the policy. The PEP must then wait for the PDPs decision, enforce the decision and execute the obligations. The purpose of these XACML obligations is mainly to add some additional actions to plain permit/deny decisions, such as logging all accesses or sending an email to the administrator if access to a certain resource is denied. That is, obligations in XACML can only be sent out as the result of an access request – spontaneously instructing a Hydra device to execute an action that has become necessary because of a change of the current situation is not possible, for example. Further, it cannot be defined who shall execute the obligation. In XACML, it is always the PEP that received the access request who has to enforce the obligation. Further, there is no way for the PDP the monitor whether the obligation was actually enforced, so the PEP has to be trusted by the PDP (which is of course always the case, in a typical XACML set-up).

With the Obligation Framework, an obligation can be triggered by any pattern of arbitrary events (e. g., sensor data, middleware events or user interactions). That is, in contrast to XACML, obligation policies are enforced asynchronously. The obligation is then sent to different obligation enforcement points (OEP) that can be located anywhere in the system (as long as they are listening to events from the Hydra Event Manager). Furthermore, it is possible to register enforcement monitors in the framework – components to monitor whether an obligation has been executed as requested or not.

So, to summarise, an obligation in XACML is different from obligation policies as supported by Hydra. Both have been designed for different purposes and they rather complement each other than competing. Below, we will also show how both features can be used to integrate the obligation policy framework with the access control framework.

	XACML	**Hydra**
Triggered By	Only access requests	Arbitrary event patterns
Location of enforcement	PEP that intercepted the access request	Arbitrary OEPs
Enforcement monitoring	No	Possible
Granularity	At policy level	Arbitrary. At rule level

Table 1: Differences between XACML and Hydra obligations

5.9 Resource Manager

The Resource Manager is responsible for representing and notifying device and operating system resources. In addition, the resource manager is responsible for tracking base and composite resources on a device. The main functionalities of the Resource Manager are to publish resource data from local devices, provide an external interface for resource management, and to interface to the operating system in order to provide information on local resources. The Device Resource Manager bases its communication with other managers entirely on Web Services.

There are three different levels of resources that are dealt with in Hydra:

- Base resources- are those that are directly provided by a system and include memory, disk space, network bandwidth etc.
- Composite resources- are composed of and use base resources and include services, clusters, total memory of a device etc.
- OS resources- the OS Interface enables manipulation and query of OS resources. For example, in Java, it would be subsumed by the Java SDK. On smaller embedded devices, it would have to be created specifically for each OS.

5.10 Hydra Storage Architecture

The Hydra Storage Architecture includes the whole storage representation in Hydra. It is needed to provide consistent interfaces to storage usable by Hydra applications. Therefore the architecture is designed to support any kind of storage, independent of the locality and of the interface. From the developers point of view it does not matter if storage is local or remotely reachable using the Hydra middleware. Storage includes persistent and non persistent storage.

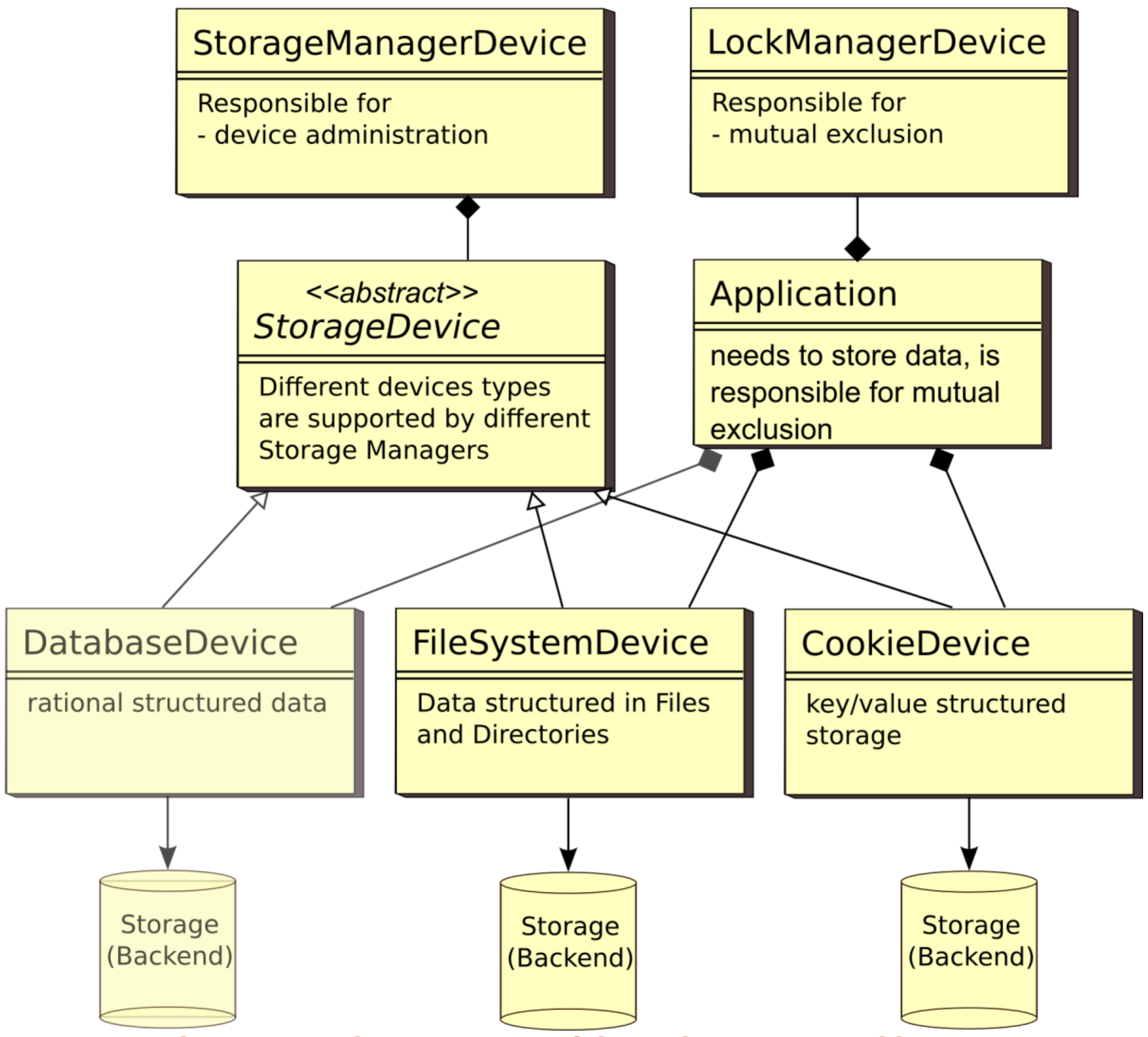


Figure 11: Basic components of the Hydra Storage Architecture

Figure 11 shows an overview over the prototype implementation of the Hydra Storage Architecture. The Storage Manager Device is responsible for administration of local Storage Devices. The Storage Devices are the representation of storage in this architecture. Persistent storage is most probably represented by File System Devices and non persistent storage can be accessed using the Cookie Device as key/value storage. However it is also possible to build persistent Cookie Devices or to build non persistent File System Devices.

The Hydra Storage Architecture is designed to be extendable by other kinds of devices, in Figure 11 this is described by the Database Device, which could be used to access rational storage in any kind of database. But devices can not only differ in the users interface, they can also differ in the way, they provide storage, e.g. there are different kinds of File System Devices for storing data local or distributed over other File System Devices.

5.11 Quality of Service (QoS) Manager

The QoS used inside Hydra is called semantic QoS or QoS for Web Services. In the Hydra project we deal with ambient, intelligent networked embedded systems. These systems typically run on resource-constrained devices, such as sensors, actuators that may sense humidity, temperature etc. The set of QoS properties Hydra deals with depends on the specific application domain. Though, it was learnt that cost as a property is very important for service requestors, whereas the power consumption of an embedded device, as a QoS property playing an important role in terms of energy efficiency, is quite important for service providers.

Service Level Performance of QoS can be measured through availability i.e. the percentage of time when the device is connected and bonded service is available, throughput i.e. the amount of data transfer ability,

& response time/ response speed which is estimated with perceived delay. Whereas Service Consumption of QoS is quantified by cost, power consumption efficiency, Service Function is measured by security, error rate, reliability and accuracy. For Hydra, as a pervasive middleware, QoS has to cover some fundamental elements, including transportation network characteristics, power and energy consumption, and underlying service properties.

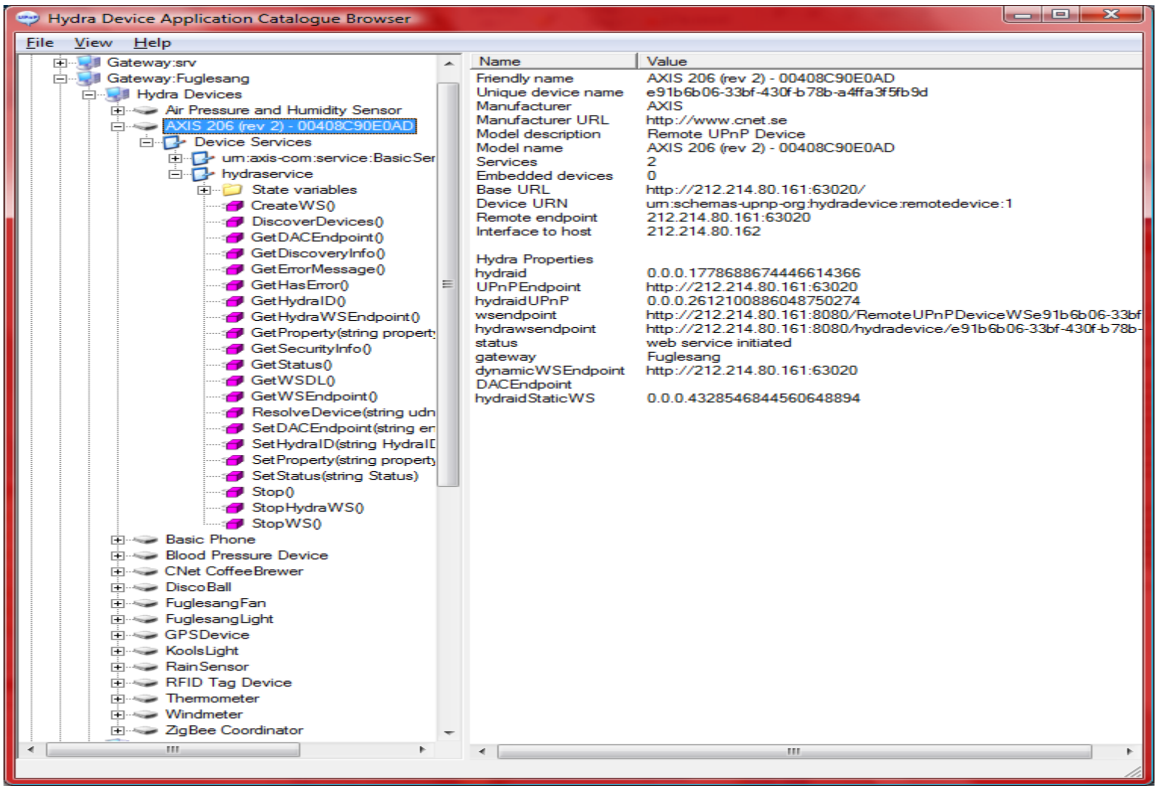
QoS ontology— is actually a set of ontology parts—formally defines the above listed important QoS parameters. Further, it contains properties for these parameters, such as its nature (dynamic or static) and the impact factor. There is also a *Relationship* concept in order to model relationships among these parameters. The Hydra QoS ontology is based on both the Amigo QoS ontology and the OWL-Q ontology. It simplifies the OWL-Q ontology in which the QoS specification idea and our listed parameters were included. The QoS ontology set is actually based on the QoSMetric ontology. It defines all the network performance parameters used to measure the quality of a network, and other parameters listed in the beginning of this section. Moreover, it defines the functions used to calculate a metric, including the *Boolean* functions, aggregation functions, and arithmetic functions. Some of the dynamic aspects, such as *Latency* or *Speed*, does not have to be expressed explicitly, but can be calculated e.g. using the SWRL rules.

The Hydra QoS Manager is a middleware component that consists of four main sub-components: Property Request/Response Handler, Rules Engine, Computation Engine, and Decision Making Engine.

5.12 Discovery Manager

The Discovery Manager is responsible for the low level physical discovery of new (existing) semantic devices using native protocol. It creates a proxy and wraps physical devices with UPnP objects and publishes them onto the network. The Application Device Manager manages all knowledge, metadata and information regarding devices that have been discovered and are active in the Hydra network. The Discovery Manager assigns a device type to the device based on Device Ontology, returns service interface for the device, handles device virtualisation (semantic devices) and semantic device aggregation, and manages the Device Application Catalogue (DAC; Figure 12). The Discovery Manager also controls a set of several Discovery Managers that are available for example, the Bluetooth Discovery Manager, the RFSwitch Discovery Manager, the SerialPort Discovery Manager, the External Discovery Manager, the UPnP Discovery Manager etc.

The Discovery Manager also adds Hydra Services to a physical device. Hydra Services (generic functions) can be in the form of Energy Services, Location Services, Storage Services, Context Services etc. The Device Application Catalogue (DAC) is a catalogue of the currently accessible devices. It provides a graphical DAC browser as a complement to the SDK/API and also provides a search interface to a DAC. Device XML is the XML structure that encodes all data and meta-data about a device. It can include properties specific to UpnP, the Hydra system and/or developer defined properties. A graphical DAC browser is used to look at the device XML.



Hydra Device Application Catalogue Browser

File View Help

```
Gateway:srv
Gateway:Fuglesang
  Hydra Devices
    Air Pressure and Humidity Sensor
    AXIS 206 (rev 2) - 00408C90E0AD
      Device Services
        urn:axis-com:service:BasicSer
        hydraservice
          State variables
          CreateWS()
          DiscoverDevices()
          GetDACEndpoint()
          GetDiscoveryInfo()
          GetErrorMessage()
          GetHasError()
          GetHydraID()
          GetHydraWSEndpoint()
          GetProperty(string property
          GetSecurityInfo()
          GetStatus()
          GetWSDL()
          GetWSEndpoint()
          ResolveDevice(string udn
          SetDACEndpoint(string en
          SetHydraID(string HydraID
          SetProperty(string property
          SetStatus(string Status)
          Stop()
          StopHydraWS()
          StopWS()
    Basic Phone
    Blood Pressure Device
    CNet CoffeeBrewer
    DiscoBall
    FuglesangFan
    FuglesangLight
    GPSDevice
    KoolsLight
    RainSensor
    RFID Tag Device
    Thermometer
    Windmeter
    ZigBee Coordinator
```

Name	Value
Friendly name	AXIS 206 (rev 2) - 00408C90E0AD
Unique device name	e91b6b06-33bf-430f-b78b-a4ffa3f5fb9d
Manufacturer	AXIS
Manufacturer URL	http://www.cnet.se
Model description	Remote UPnP Device
Model name	AXIS 206 (rev 2) - 00408C90E0AD
Services	2
Embedded devices	0
Base URL	http://212.214.80.161:63020/
Device URN	urn:schemas-upnp-org:hydradevice:remotedevice:1
Remote endpoint	212.214.80.161:63020
Interface to host	212.214.80.162
Hydra Properties	
hydraid	0.0.0.1778688674446614366
UPnPEndpoint	http://212.214.80.161:63020
hydraidUPnP	0.0.0.2612100886048750274
wsendpoint	http://212.214.80.161:8080/RemoteUPnPDeviceWSe91b6b06-33bf
hydrawsendpoint	http://212.214.80.161:8080/hydradevice/e91b6b06-33bf-430f-b78b-
status	web service initiated
gateway	Fuglesang
dynamicWSEndpoint	http://212.214.80.161:63020
DACEndpoint	
hydraidStaticWS	0.0.0.4328546844560648894

Figure 12: Device Application Catalogue – Graphical View

5.13 Security Library

This chapter deals with the mechanisms to protect messages which are exchanged between Hydra Managers. It also describes managers which are used to support the security mechanisms and which also provides functionalities to the Hydra developers, who want to use cryptographic functions in their Hydra applications.

Core Hydra

A single Hydra-enabled device can be distributed across many physical machines. In that case, internal parts of this Hydra device communicate over a public network, e.g. the Internet. To protect this kind of internal communication, the Core Hydra security mechanisms are in place. Core Hydra security is based on symmetric keys that have to be generated by the device developer and must be deployed to each manager of the Hydra device. The implementation of the Core Hydra module is integrated into the Network Manager in the form of an Axis handler.

Inside Hydra

The Inside Hydra approach is based on public and private key pairs. Therefore, other considerations as in Core Hydra come into effect. Issues have to be solved for validating certificates, handling tokens, verifying signatures and token creation, storing and deletion. Due to the fact that a HID can change, we consider generating certificates on the fly, which then will be exchanged among the communication partners. The dedicated TrustManager is engaged in the verification of these generated certificates. The deletion of such a certificate is also due to the act that an HID of a counterpart is not longer valid.

5.14 Security Design

To secure messages between Network Managers over the insecure network, we use the Web Service Security (WSS) to secure messages (Strings), similar to Core Hydra. The main differences are the handling of security tokens (i.e., X509 certificates or other tokens used by other trust models).

Another aim of this Security Library is also to provide an interface for different security mechanisms. The developer will have the possibility to use any library which supports our envisaged interface; this can also mean that the developer can, for example, plug in a different security library, which suits her/his needs.

The integrity and non repudiation of messages will be achieved by the use of signatures. Due to the fact that HIDs can change and communication Partners (Network Managers) are previously unknown to each other, considerations like token generation and token exchanging must be taken into account.

The dedicated Trust Manager is used to verify the integrity of these tokens by verifying the certificates which are used by the Network Managers. A trust level can be set up by the developer which will be used to evaluate the trustworthiness of a generated and used certificate. Also different trust models can be used, which are also configured by the developer. Trust models can make use of Public Key Infrastructures, of Web of Trust mechanisms, or even an interaction by the user.

Every time a new communication is initialised the Network Manager generates its own certificate which will be shared via a handshake mechanism, described below, between the different Partners. For the time of communication they will be stored in each involved manager's key store. Certificates can also be deleted after a certain time, i.e. when a HID is dropped or is not longer valid. That means for every HID there will be a generated public/private key pair.

6. Hydra Tools

6.1 Device Application Catalogue Browser

The DAC is the interface to the Application Device Manager and shows all discovered devices in a Hydra network with relevant information on how to access them.

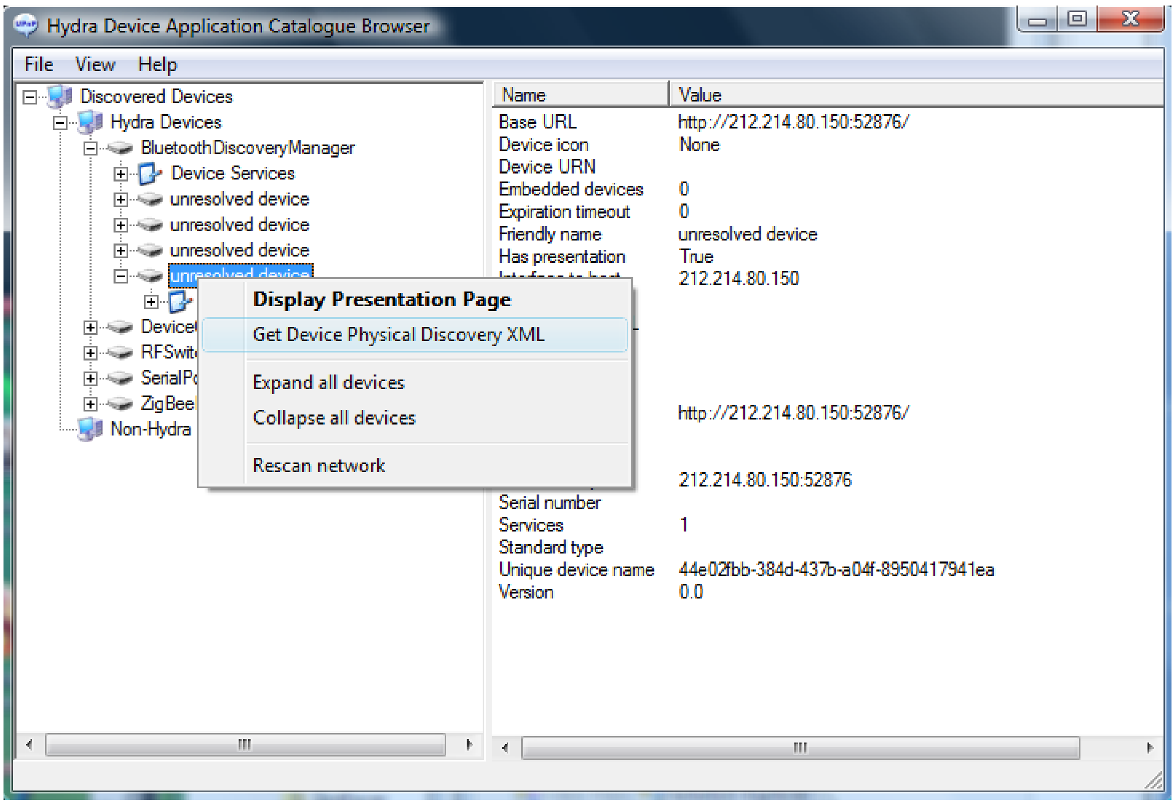
6.1.1 The Graphical Browser

A fundamental part in every Hydra-based application is the Device Application Catalogue (DAC, Figure 13), which is managed by the Application Device Manager. This is a runtime component that keeps track of and manages all devices that are currently active within an application. The Hydra Device Application Catalogue serves all Hydra middleware managers with the information and metadata they need regarding devices, their services, and their status.

Hydra uses the Hydra Device Ontology and models for discovery to recognise new devices when they enter into a Hydra network. Based on the discovery model it queries the Device Ontology to deduce what type of device has entered the network. The Hydra network can be queried by different middleware managers to retrieve a service interface for different devices.

A Hydra browser has been developed to allow a user/developer to graphically browse the Hydra network and inspect properties and services of devices. The browser tool also allows the user to invoke the different services offered by devices. By manually invoking the different services we can also actually illustrate the role the Device Application Catalogue plays in the Hydra middleware.

Each Discovery Manager keeps track of the device it has discovered and tries to elicit as much information as possible from the device. All this physical discovery information can be accessed by calling the service "Get Device Physical Discovery XMl".



Name	Value
Base URL	http://212.214.80.150:52876/
Device icon	None
Device URN	
Embedded devices	0
Expiration timeout	0
Friendly name	unresolved device
Has presentation	True
Interface to host	212.214.80.150
	http://212.214.80.150:52876/
	212.214.80.150:52876
Serial number	
Services	1
Standard type	
Unique device name	44e02fbb-384d-437b-a04f-8950417941ea
Version	0.0

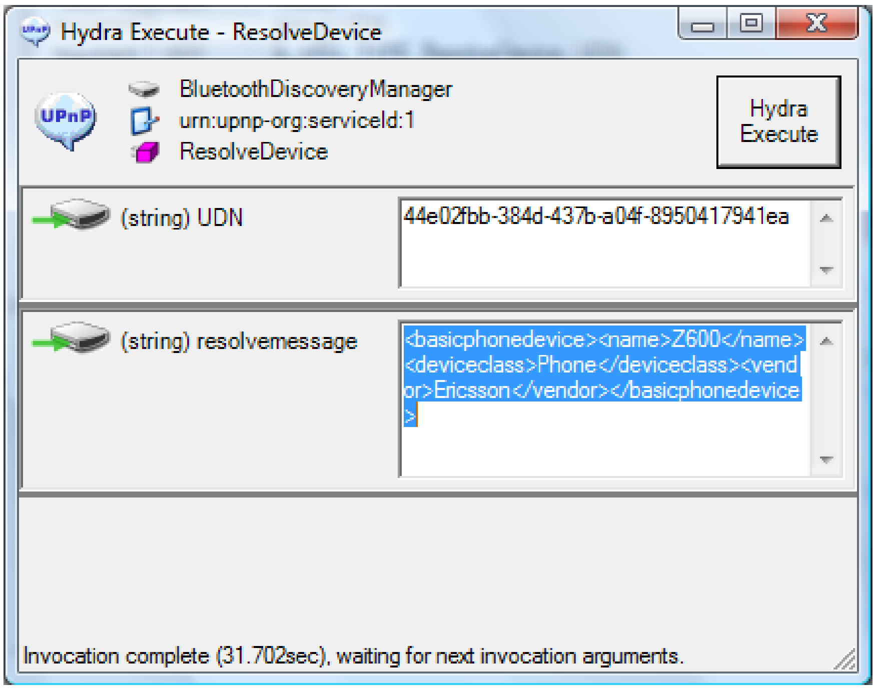
Figure 13: The Hydra DAC Browser and retrieving discovery information

This discovery information is returned as an XML document, which can be seen in Figure 14 above:

In Figure 14 we can see that it is a Bluetooth Device that has been discovered, it has the Bluetooth Major DeviceType "Phone" and Minor DeviceType "CellPhonePhone" (Major DeviceType and Minor DeviceType are part of the Bluetooth standard.

The Bluetooth Discovery Manager has also managed to extract the different Bluetooth services offered by the device. This discovery information can now be used to reason about what type of device has been discovered. The physical discovery XML is given to the Device Ontology which deducts that this device corresponds to a "Basic Phone" in the Hydra Device Ontology.

By invoking the service "Resolve Device" we can now tell the Bluetooth Discovery Manager that this is a "Basic Phone". The idea is of course to do this programmatically, but here we do it manually for illustration purposes.



(string) UDN: 44e02fbb-384d-437b-a04f-8950417941ea

(string) resolvemessage: <basicphonedevice><name>Z600</name><deviceclass>Phone</deviceclass><vendor>Ericsson</vendor></basicphonedevice>

Invocation complete (31.702sec), waiting for next invocation arguments.

Figure 14: Resolved information is sent as an XML structure to the Discovery Manager

The Discovery Manager then creates and publishes the Device to the network as a "Basic Phone" device. The Basic Phone device is now available together with the services offered by a Basic Phone (in this case a set of SMS read/send functions). These services can be invoked from the Browser, and can, for instance, send an SMS.

To summarise, the Discovery process follows this sequence:

Hydra Application Device Catalogue → Hydra Device Identified → Device Application Catalogue → New HYDRA Device → Lookup Device → Device Device announcement → Application Device Manager

As a device is discovered in the network, its type is resolved against the device ontology, and then entered into the HYDRA application. We can also use the Browser to retrieve a service description for a web service that allows us to access the device programmatically.

6.2 Hydra IDE

The Hydra IDE is the development environment designed in Hydra, to allow developers to easily use the various components of the Hydra Middleware, that are integrated into a single environment. The Hydra IDE components are based on the Eclipse rich client platform (RCP), which allow them to naturally integrate into the traditional Eclipse IDE which most software Java developers know and are used to and so the usability of Eclipse-based components is much better than that of a proprietary IDE.

The Hydra IDE provides functionality for device developers as well for application developers and administrators. For all use cases, it is sensible to expect that the IDE will not necessarily run on the same machine as the middleware instance itself. Thus, the Hydra IDE provides two ways of connecting to a remote Hydra instance from which the developer can access: via a local NetworkManager and via Remote OSGi (R-OSGi). The remote connection feature enables the Hydra IDE to develop software for Hydra middleware without having a Hydra installation running locally, connecting to a remote working Hydra installation.

More details about the Hydra IDE itself, and how to use the various components to create applications utilising the Hydra Middleware, are given in an upcoming deliverable - D12.9 *Final External developers workshops teaching materials* - that is targeted at developers and users of the Hydra IDE.

6.3 Limbo

Limbo is a web service compiler for embedded devices. It helps developers create web services by generating the basic code necessary for realising a webservice on a given target platform. The code generation is guided by a declarative specification of the features required in the web service. Further, it can help implement the generated services so they emit state changes through the EventManager. This enables Flamenco to provide self-management algorithms with the information they need about the managed system's state. This chapter will give a general high level overview of the updated Limbo features and its compilation process. In addition, an overview of how to use these new features and the generated artefacts will be discussed.

The Limbo feature and compilation process is shown in Figure 15 and Figure 16 in which the consideration of different SOAP transportation protocols are variants in the feature diagram and are also reflected in the compilation process.

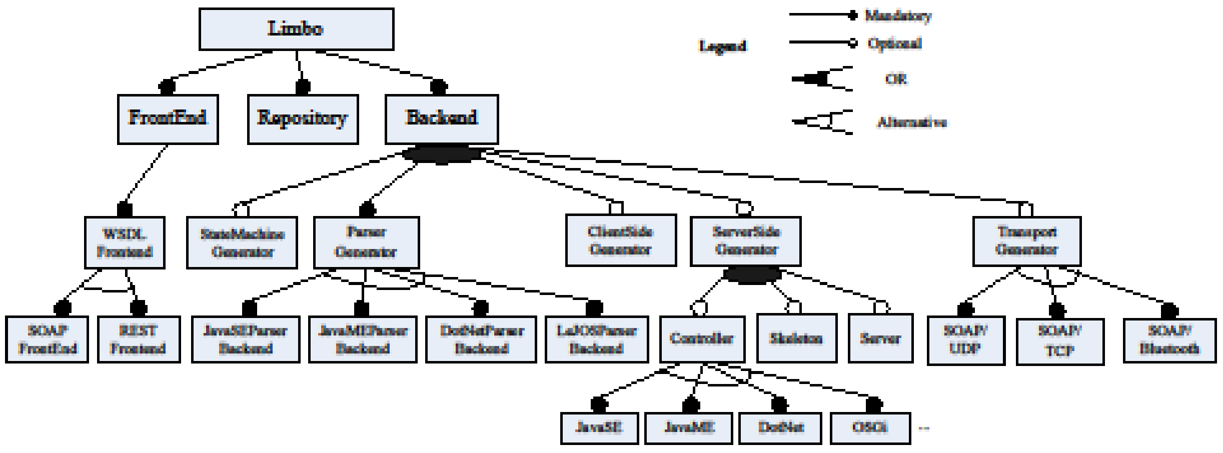


Figure 15: Limbo feature diagram

The implementing web services on networked, embedded devices used in Hydra leads to a set of challenges, including productivity of development, efficiency of web services, and handling of variability and dependencies of hardware and software platforms.

To address these challenges, a web service compiler called 'Limbo' was developed, in which Web Ontology Language (OWL) ontologies are used to make the Limbo compiler aware of its compilation context such as device hardware and software details, platform dependencies, and resource/power consumption.

The ontologies are used to configure Limbo for generating resource-efficient web service code. The architecture of Limbo follows the Blackboard architectural style and Limbo is implemented using the OSGi Declarative Services component model. The component model provides high flexibility for adding new compilation features. A number of evaluations show that the Limbo compiler is successful in terms of performance, completeness, and usability.

Web services promise interoperable, composable, and reusable services in architectures in which Service-Oriented Architecture principles are employed. Increasingly, and from a deployment viewpoint, such architectures also include resource-constrained embedded devices.

Development of embedded web services must handle variability of hardware and software, and possible dependencies between hardware and software. This is particularly true when a different implementation language and communication protocols are involved. Limbo supports developers of pervasive web service applications in order to improve productivity.

Code generation is an effective way to improve reuse and to improve the productivity. To support this, an OSGi-based and ontology-enabled web service compiler called Limbo to generate resource-efficient code was developed in the scope of the Hydra Middleware project.

The idea behind Limbo is: using OWL ontologies to encode device hardware and software details, including their dependencies, which can then be used as compiling contexts during code generation for a specific device, and the Blackboard architectural style realised through the OSGi platform.

This OSGi-based implementation of Blackboard architecture make the adding of compiling for different targeted platform dynamic without affecting existing components.

In this way, the developer is liberated from the understanding details and dependencies of various platforms and at the same time can generate resource efficient code according to requirements and the details of a device.

The Limbo compiler is flexible in terms of e.g. supporting different type of communication protocols, service discovery protocols, and also programming platforms.

WSDL (Web Services Description Language) is an XMLbased language for describing Web services and how to access them, and should be used as central point for web service generation. Such files are provided as input to Limbo and Limbo follows the "Blackboard" architectural patterns [22] in which a central Repository stores data (initially WSDL data) related to the transformation process and on which Frontends and Backends operate to read and write information.

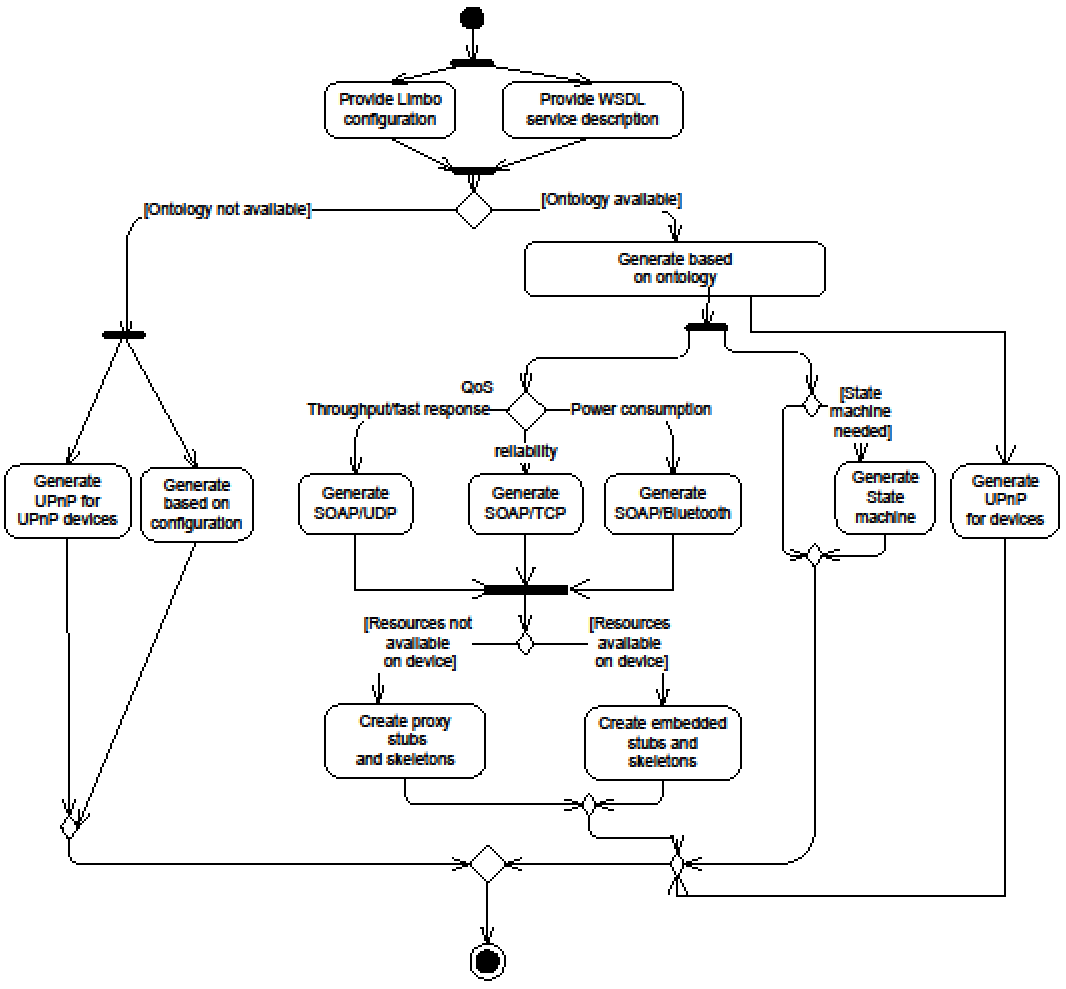
An essential feature is the parser backend with different implementation languages such as Java SE/Java ME (Java Standard Edition/ Java Micro Edition). There are also requirements for the generation of client-side stubs and/or server-side skeletons, and for the transport code for network communication between client and a server. To provide the possibility of handling dynamicity of device state changes, a state machine backend is designed to generate state machine stubs. At runtime, when executing the service, these state machines report relevant state changes of the device.

To provide the possibility of finding devices supporting different communication protocol, service discovery backend is also developed. UDDI [20] is designed for wired network for service discovery, which is not suitable for pervasive computing environment. For the Hydra middleware, UPnP was chosen as the main protocol for service discovery, therefore, an UPnP backend, which is used to generate the UPnP device description, and the service description, has been implemented, in order to make both devices and services available in the UPnP discovery process.

This makes use of an ontology description of the device and service. Originally WSDL has nothing to do with (OWL) ontologies. Recently, there have however been a number of efforts to link OWL with WSDL, for example OWL-S [18] and SAWSDL [19].

OWL-S is targeting automatic selection, composition, and execution of web services, which is quite different from what is required for linking WSDL with web service generation. Also OWL-S is a heavy-weight solution in that it requires a major shift from WSDL to describe semantic services. SAWSDL is light-weight and links the service defined in a WSDL file to external sources of semantic description in OWL.

This work inspired us to extend the semantics of WSDL binding, through a *hydra: binding*, in order to make use of ontologies during compilation. This new binding must refer to a device instance in the Device ontology (which imports other ontologies). The Limbo ontology frontend will resolve the URI and retrieve thermometer hardware and software information.



Provide Limbo configuration
Provide WSDL service description
[Ontology not available]
[Ontology available]
Generate based on ontology
QoS
Throughput/fast response
Power consumption
reliability
[State machine needed]
Generate UPnP for UPnP devices
Generate based on configuration
Generate SOAP/UDP
Generate SOAP/TCP
Generate SOAP/Bluetooth
Generate State machine
Generate UPnP for devices
[Resources not available on device]
[Resources available on device]
Create proxy stubs and skeletons
Create embedded stubs and skeletons

Figure 16: Limbo compilation process

Figure 16 shows the compilation process of the Limbo compiler. The following steps are involved:

- Provide WSDL service description: The main input for Limbo is a WSDL file, and Limbo also supports that WSDL files reference the Hydra device ontology.

- Generation based on configuration or ontology. If an ontology instance for the device is available and referenced, a device specific platform information will be used to generate client and/or server code. Otherwise, generation configuration is based solely on developer supplied parameters.

- Create embedded/proxy stubs and skeletons. Stubs and skeletons for the device service are created according to the device capability. If there are not (enough) resources for running code directly on devices, proxy code is generated based on OSGi. For the thermometer, as it does not have any computing capability itself according to the retrieved platform information from the ontology, proxy code will be generated. Currently, Limbo supports Java SE and Java ME code generation.

- Generate device state machine stubs. If a device state machine instance is available in the StateMachine ontology which is imported by the Device ontology, and Limbo is configured to generate state machines, then state machine stubs will be generated for a device.

- Generate a UPnP device description and service description. If a device supports UPnP, then the generation will be based on the default UPnP information provided by the device manufacturers, or else the device description will be based on the information in the Device ontology, and UPnP service description will be generated based on WSDL description.

OSGi provides a service-oriented component-based platform for use by systems that require dynamic updates, and minimal disruptions to the running environment. This provides us with more flexibility than the Blackboard architecture itself by allowing the dynamic addition of new compilation features, for example, the supporting of new service discovery protocols, and new requirements for generating code for different platforms.

A Limbo plug-in is a normal OSGi bundle and as such needs to implement the *BundleActivator* interface of OSGi. This interface has two methods, start and stop, which are used to start and stop the bundle respectively.

The UPnP component is an example of a plug-in. The component creates device services that are discoverable by the UPnP protocol. The creation involves the generation of a UPnP device and service descriptions in XML syntax. The generated code makes these descriptions including the devices' capabilities available on the network, so other devices can learn about the device.

The version based on OSGi-DS is much more flexible and extensible, where no service tracking for backends and frontends are needed and can be added dynamically, and will be validated by a configurator executing SWRL rules to validate the configurations.

The use of ontologies makes the tool capable of handling different type of devices, and Limbo can be easily extended to support other service discovery protocols than UPnP. XML Screamer [21] is an example of a tool that generates specific XML parsers for a specific XML Schema. Limbo is clearly related to this in the way that the generation of specific parsers for SOAP XML data is described by a WSDL files (that may also contain an XML Schema specification).

Limbo has a highly flexible architecture which can be easily extended with the generation of code for .NET code, and other specialised platform, with the support of ontologies and rule languages to specify and regulate the generation processes.

6.4 Flamenco

The Diagnostics Manager is the central part of the Hydra middleware that has a responsibility to support self-management. Hydra implements a service-oriented architecture based on web service interaction among devices. Thus a reasonable granularity to build a self-management system at the level of web service requests and responses was needed. Concerning web services, a significant body of existing research in self-managing systems assumes the system under management to be service oriented. Thus, using web services ensures that this knowledge can be leveraged in systems based on the Hydra architecture. Furthermore, we are interested in the states of devices per se, i.e., is the device operational, stopped, not working and if it is operational what is the value of its sensor readings (if any) or its actuator state (if any).

This leads us initially to focus on status reporting of the following two forms:

State change reporting: The Limbo web service generation tool supports the generation of state machines describing the states of a device and its associated services. These state machines report their state changes as events through the Hydra Event Manager.

Web service request/reply reporting: The interaction among devices and managers in Hydra takes place via web service calls. The requests and replies (and their associated data) can subsequently be used to analyse the runtime structure of running Hydra systems. To do this, we have implemented an IP sniffer that is able to report about TCP/IP packets being sent between hosts.

An integral part of Flamenco then becomes to make sense of system level events ("Status") and transform that into representations of, e.g., architectural components and connectors.

6.4.1 Flamenco CPN

Flamenco/CPN generalises this to use a state-based approach, more specifically a Petri Net-based approach, to interpret system events as architectural events and to reason upon these elements. Coloured Petri Nets is a formal, graphical modelling language with well-defined syntax and semantics.

Flamenco/CPN must be realised in the Hydra middleware and implemented using CPN Tools. This is to a certain extent achieved by coupling Flamenco/CPN to the publish/subscribe subsystem of Hydra (i.e., the Event Manager) and taking advantage of that Limbo-generated services already use the Event Manager to publish state information.

6.5 An OWL/SWRL-Based Approach – Flamenco/SW

Diagnosis is a complex task which needs intelligence to infer what the reason for error and its consequence is. Ontologies are extensively used in pervasive computing for achieving context-awareness and as already mentioned they are also used within the Hydra project. This naturally led to the exploration of an approach that can be built on existing ontologies used in Hydra.

Flamenco uses the Ontology Web Language, OWL, as well as the Semantic Web Rule Language, SWRL. OWL provides a way to represent knowledge about a system, while SWRL provides complementing reasoning capabilities. These capabilities are needed to enable Flamenco to model a running system and reason about the knowledge it gathers to, for example, realise self-optimisation. These particular technologies for representation and reasoning were chosen to match the use of semantic technology in other parts of the project.

The current design of the Diagnostics Manager will cover all the components of Fault Detection, Device Monitoring, Communication Monitoring, Device Status, and Log facility.

Log facility is now running in a dynamic way in the state machine ontology, where three historical results will be kept, as well as one current state and its activity result. Communication monitoring is conducted with mainly the IPSniffer, which reports all the web service calls including the process id, invoking time, source target IP address and port number. Device status and device monitoring is fulfilled with the state machine ontology and event mechanism.

Generally, the design of the OWL/SWRL based Diagnostics Manager is following the layered architecture. The bottom of the architecture is the ontologies/rules, in which knowledge of devices, and state based diagnosis are encoded and will be used when there are state changes. When there are state change events, the device state machine instance in the state machine ontology needs to be updated, and also these state changes will be published with state machine state changes as an event topic. While the diagnostics manager will also serve as an event subscriber to the state machine state change events, it will then update the corresponding state instances in the ontology. At the same time, this will trigger the diagnosis of the device status, executing the SWRL rules to monitor the health status of devices, and also trigger the reasoning of possible device errors and their resolutions. At the same time, the diagnostics manager will publish the diagnosis results as an event publisher.

It may also be noted that WSDL has been extended to have SA-WSDL-like external references in bindings in order to reference OWL device descriptions.

7. Summary

As shown in this document, Hydra uses a variety of established standards and has been developed using the latest technologies in the fields of middleware, programming and device communication. In this way the flexibility and interoperability of the Hydra Middleware is assured through its programming interfaces which can be used by commercial companies as well as by the Open Source community to produce and develop applications for network-embedded devices in a heterogeneous environment.

The use of technologies, such as Java, Eclipse and P2P networks, which are very popular, by manufacturers and developers around the globe, is an advantage when developing rapid prototypes as well as integrated services and applications; particularly given the usual pressures of restricted time and human resources.

Hydra is also adapting and changing concepts and technologies, as well as introducing new ideas and implementations for certain sub-problems, and making such features available within the Hydra SDKs and IDEs.

The capability of device discovery supported by an ontology also reflects the strength and variability of the Hydra Middleware.

Clearly defined and structured APIs and usable and supportive developer tools in all parts of the middleware are also the base for a highly user and developer friendly software product which is easily expandable because of its distributed architectural design.

The use of OS-independent languages and tools (e.g. Java, Eclipse) ensures that Hydra can be used in a vast variety of systems and devices.

Device Developer Tools are provided to ensure a flawless integration into the middleware and the final applications.

Information technology is an omnipresent Partner in the world of today and is a necessity for nearly all people in their workplaces. Over the past decade there has also been the rise of computer risks and information frauds which came with the rapidly expanding success of the Internet. With a clear security goal in mind, Hydra enables and ensures the authenticity, the privacy and integrity of the data and applications in the scope of Hydra. Therefore a bundle of security mechanisms are built in with additional functionalities and tools used either during the application development process or while configuring the necessary changes in the deployed application by the developer and/or user.

The use of Hydra has already been demonstrated in several domains. The three example domains, Building Automation, Health Care and Agriculture have proved that the Hydra middleware is highly adaptable to different types of devices, systems and protocols.

The use of interface definitions also ensures a fast integration of new protocols or device types into Hydra.

The Open Source strategy undertaken by the Hydra Consortium supports the longer term usability of Hydra and provides small and medium sized enterprises as well as other users with an out-of-the-box solution for developing applications for heterogeneous devices in a distributed architecture.

8. Glossary

This chapter aims at providing a comprehensive understanding of important terms used in and derived from the Hydra project. In addition, the terms listed below try to convey a sense of their application and present the background of the fundamental concepts. Even if some of the subsections seem to be a repetition of things already documented, this chapter can be seen as a central point of access to a description of the Hydra terms. The definitions listed here have been agreed on by the Hydra Consortium. (The terms are ordered from high-level to low-level)

Physical Device:

A "Physical Device" is a common device that offers some functions that affect the "physical world".

Such functions could for example be providing light, heat, wind, open door, or reports physical properties such as temperature, blood pressure, pulse, movements, etc. Hydra constitutes a middleware that enables networking of physical devices.

Appliance:

An "Appliance" represents a physical device that is dedicated to a single purpose. Appliances refer to more complex physical devices and are especially prominent in the field of home automation or home entertainment.

Hydra-Compliant Physical Device:

A "Hydra-compliant Physical Device" is a physical device that can be Hydra-enabled. Hydra-compliant physical devices divide into 5 different classes (see Section 3) that determine the procedure to be used to Hydra-enable devices and integrate them into a Hydra network. As a baseline capability such devices need to offer some external interface for communication and control.

Examples of such external interfaces supported by the Hydra middleware are Bluetooth, ZigBee, RF, RFID, serial ports, USB, etc.

Hydra-Enabling a Device:

"Hydra-enabling a Device" means the process of making the functions of a Hydra-compliant physical device available and controllable for other devices in a Hydra network. Depending on its device class, three methods make such a device Hydra-enabled:

- Installing (parts of) the Hydra middleware on the device
- Using the Limbo tools to embed Web Services on the device and generate a Proxy
- Using a Proxy to represent the device on a Gateway

At the end of this process the functions of this device can be invoked using Web Services, and metadata about the device is provided in the format and protocol required by Hydra.

Hydra-Enabled Device:

A "Hydra-Enabled Device" is a Hydra-compliant physical device that has successfully run through the Hydra-enabling process. A Hydra-enabled device owns a software representation, i.e. a Hydra Device, in a Hydra network and

- Can be discovered by other devices in a Hydra network
- Makes all or a subset of its functions accessible as Web Services
- Offers its Web Services either natively (embedded code) or through a proxy
- Supports UPnP and advertises its entry into a Local Area Network through UPnP broadcasting
- Supports Hydra Generic Services and Hydra Energy Service.

Hydra Device:

A "Hydra Device" constitutes the software representation of a Hydra-enabled device and its functionalities, in order to enable access and control. The Hydra Device can either run as a Proxy for the Hydra-compliant physical device on a gateway or it can run embedded in the device. A Hydra Device can obtain Hydra identifiers for its services (HID) and also application specific identifiers. Furthermore, a Hydra Device implements the "Hydra Generic Services" and "Hydra Energy Services". For one physical device there might exist one or more Hydra Devices. A Hydra Device might also incorporate services from several physical devices.

Semantic Device:

A "Semantic Device" represents a composition of one or more Hydra devices and constitutes an SDK construct. A semantic device is dynamically bound to its Hydra devices at runtime. Therefore a semantic device might only be partially instantiated at runtime. A semantic device is discoverable in the same way as and also acts as any Hydra Device. The description of the semantic device is part of the Hydra Device Ontology.

Gateway:

A "Gateway" is a physical device with IP capabilities, which manages a set of proxies for controlling Hydra devices. A gateway must support Web Services and UPnP and should also be able to run Hydra Discovery Managers. In addition, a gateway may also host other components of the Hydra middleware.

Proxy:

A "Proxy" is a Hydra Device that consists of a software component responsible of communicating with a physical device, understanding the technology used and the format of the data exchanged. It is deployed on a gateway and represents the device to be controlled.

Bridge:

A "Bridge" represents a software component that resides inside a Gateway and translates any non-IP communication into an IP based communication. It is used by Hydra-enabled devices with non-IP capabilities to communicate inside the Hydra network.

Hydra Network:

A "Hydra Network" represents a network of Hydra Devices and applications that communicate with each other using Web Services and IP communication on top of a Peer-to-Peer overlay.

Hydra Middleware:

The "Hydra Middleware" is a collection of interrelated components, i.e. Hydra Managers, that work together to realise a platform of networked heterogeneous physical devices. The Hydra middleware allows such devices to be part of an ambient intelligence environment.

Device Discovery:

The process "Device Discovery" covers several steps where a physical device is discovered, semantically resolved and made accessible as a Hydra Device. In order for a device to be discovered in a Hydra network, a definition of the device type must exist in the Hydra Device Ontology.

Hydra Manager:

A "Hydra manager" (or short "manager") constitutes the major building blocks that make up the Hydra middleware. A Hydra manager encapsulates a set of operations and data that realise a specific functionality and is mostly subdivided into several internal components.

Hydra Generic Services:

The Hydra Generic Services are supported by all Hydra Devices and contain a set of meta-data methods that can be used to query the device about its properties.

Hydra Energy Services:

The Hydra Energy Services are supported by all Hydra Devices and provide methods to retrieve information from the energy profile of the device and from the energy policy.

Hydra Identifier (HID):

A "Hydra identifier" (or simply "Hydra ID" or shorter "HID") constitutes a unique identifier for every Hydra Device, service or resource within a Hydra network. The Network Manager generates the HID, and is responsible for the matching between logical and physical identifiers and for the propagation of this information to other peers of the Hydra network.

CryptoHID:

An application developer has the opportunity to assign his own CryptoHID to a certain Hydra Device. This CryptoHID can directly be used throughout the application code and referred to when expressing security, energy and other policies.

Session:

A "Session" traces the communication between elements of a Hydra network, in order to keep the communication coherent. Sessions allow the maintenance of the state of each network element as they communicate with each other. The Network Manager comprises a dedicated Session Manager that creates and maintains the lifecycles of the session objects.

Ontology:

An "Ontology" is a representation of the knowledge of a formally defined system of concepts and relations. In addition, an ontology can contain inference to derive new knowledge and integrity rules to assure its validity. Therefore, an ontology forms a network of information and logical relations described through a formal language such as the Web Ontology Language (OWL).

Hydra Device Ontology:

The "Hydra Device Ontology" is an ontology that contains knowledge about device classes, their properties and services offered.

Device Model:

A "Device Model" describes the properties and services that a certain device class offers. The Device Model is expressed using the SCPD XML format of the UPnP standard.

Hydra Peer-to-Peer Architecture:

The "Hydra Peer-to-Peer Architecture" allows Hydra Devices in different local Hydra networks to access and communicate with each other. This means Web Services calls can be executed remotely over a P2P overlay.

9. References and further Reading

[1]	Brinkmann, A., Effert, S., and Gao, Y. (2009). D3.12 Updated Grid Architecture Report. Technical Report, University of Paderborn.
[2]	Ingstrup, M., and Zhang, W. (2008). D4.8 Self-star properties DDK prototype and report. Technical report, UAAR.
[3]	Al-Akkad, A.-A., Kostelnik, P., and Zhang, W. (2009). D4.10 Quality-of-service enabled hydra middleware. Technical report, Fraunhofer FIT.
[4]	Drools – Business Rules Management System. <http://labs.jboss.com/drools>
[5]	eXist-db Open Source Native XML Database. <http://exist.sourceforge.net>
[6]	OASIS - http://www.oasis-open.org
[7]	Sun XACML Implémentation. <http ://sunxacml.sourceforge.net>
[8]	eXtensible Access Control Markup Language (XACML) Version 2.0 (2005) - http://www.oasis-open.org/committees/xacml/
[9]	IBM, Web Services Security http://www.ibm.com/developerworks/library/specification/ws-secure
[10]	W3C, XML Security Introduction http://www.w3.org/2004/Talks/0520-hh-xmlsec/slide4-0.html
[11]	http;//www.osgi.org/Links/DeveloperKits
[12]	The SENSORIA Development Environment, CASE Tool for SOA Development http://home.mit.bme.hu/~rath/ppt/SDE.pdf
[13]	http://msdn.microsoft.com/en-us/netframework/aa904594.aspx
[14]	http://www.mono-project.com
[15]	http://www.oscaf.org/nrl_ontolog]
[16]	http://protege.stanford.edu/plugins/owl/jena-integration.html http://protege.stanford.edu/plugins/owl/api/guide.html
[17]	http://msdn.microsoft.com/en-us/netframework/aa904594.aspx
[18]	http://www.w3.org/Submission/OWL-S
[19]	http://www.w3.org/2002/ws/sawsdl/
[20]	http://www.uddi.org/
[21]	M. Kostoulas, M. Matsa, and N. e. a. Mendelsohn. XML screamer: an integrated approach to high performance XML parsing, validation and deserialization. 15th international conference on World Wide Web, pages 93–102, 2006.
[22]	M. Shaw. Some Patterns for Software Architectures. Pattern Languages of Program Design, 2:255–269, 1996.

Further information about the Hydra Middleware Project

Hydra Website	http://www.hydramiddleware.eu
Hydra Public Deliverables	http://www.hydramiddleware.eu/articles.php?article_id=90
Hydra in Wikipedia	http://en.wikipedia.org/wiki/Hydra_Project_%28EU_Project%29
CNet Demo	http://hydra.cnet.se