Contract No. IST 2005-034891

Hydra

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

D4.10 Quality-of-Service Enabled Hydra Middleware

Integrated Project
SO 2.5.3 Embedded systems

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

Published by the Hydra Consortium 2009-12-18 - version 1.2
Coordinating Partner: Fraunhofer FIT

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

Dissemination Level: PU

Document file:	D4.10 Quality-of-Service Enabled HYDRA Middleware.docx
Work package:	[WP4 – Embedded AmI Architecture]
Task:	[T4.5 – Quality-of-Service]
Document owner:	[Ahmad-Amr Al-Akkad (FIT)]

Document history:

Version	Author(s)	Date	Changes made
0.1	Ahmad-Amr Al-Akkad (FIT)	28-05-2009	Table of Contents
0.2	Ahmad-Amr Al-Akkad (FIT)	17-06-2009	Draft section 2
0.3	Ahmad-Amr Al-Akkad (FIT)	19-06-2009	Draft section 3
0.4	Ahmad-Amr Al-Akkad (FIT)	22-06-2009	Draft section 5.2
0.5	Ahmad-Amr Al-Akkad (FIT)	23-06-2009	Draft section 5.3
0.6	Peter Kostelnik (TUK)	25-06-2009	Draft section 3.3.1
0.6	Ahmad-Amr Al-Akkad (FIT)	25-06-2009	Draft section 6
0.7	Weishan Zhang (UAAR)	26-06-2009	Draft section 4.2
0.7	Ahmad-Amr Al-Akkad (FIT)	26-05-2009	Integration of section drafts
0.8	Ahmad-Amr Al-Akkad (FIT)	29-06-2009	1st frozen version for proof reading
0.9	Ahmad-Amr Al-Akkad (FIT)	02-07-2009	Changing structure, section 2.4.3
1.0	Ahmad-Amr Al-Akkad (FIT)	02-07-2009	2nd frozen version for proof reading
1.1	Ahmad-Amr Al-Akkad (FIT)	03-07-2009	Draft version for internal review
1.2	Ahmad-Amr Al-Akkad (FIT)	05-07-2009	Final version to be submitted to EC

Internal review history:

Reviewed by	Date	Comments
Mads Ingstrup (UAAR)	30-06-2009	Related Work for Semantic Service Selection missing, Structure, and Typos and formal Issues. Approved with Comments
Francisco Milagro (TID)	03-07-2009	Only several typos errors and minor comments. Accepted

Index:

Figures:

Tables:

Executive summary

This written report documents, for a developer audience, the work on WP4 QoS related middleware components that has been implemented for the third iteration of the Hydra project. In particular, this reports keeps records on the 'QoS Manager' subtask of T4.4 and completely on T4.5 (for details see Hydra description of work [1]).

The WP4 QoS related tools and middleware components have been extended for introducing new functionalities, changed and refined according to lessons learned from the second year prototype for providing improved functionalities:

- Ontology Manager provides a procedure to describe semantically Hydra services with QoS properties.
- Ontology Manager provides an accessible web interface for retrieving data values of QoS properties of Hydra services and the (embedded) devices on which these services are running.
- QoS Manager provides an accessible web interface for retrieving the best-suitable services out of a range of Hydra services with same functionality, though they differ in terms of QoS. These services are the result of semantic service selection.
- QoS Manager also provides an accessible web interface for monitoring quality "views of services" that are built according to a set of specific service parameters. The monitoring covers static and dynamic QoS properties as well. For instance, the self-management component regularly requires an update of QoS properties.

Basically, for developers this document explains the required steps for enabling Hydra services with QoS properties, and as well how to consume Hydra services being aware of QoS properties. From a technical point of view, these steps comprise:

- Defining concepts (, or also called classes) which describe several QoS properties in Hydra service and device ontology.
- To invoke the reasoning over concepts in Hydra service and device ontology for retrieving a specific set of data values of QoS properties. As a prerequisite, the reasoning requires as input queries. For instance, a query may be written in SPARQL [2].
- Agree on an application specific computation for several values of specific QoS properties; the computation can be simple or rather complex depending on the domain, QoS properties, defined units for QoS properties etc.
- To perform semantic service selection resulting the best-suitable services for a certain task. The result may take as basis the average fitting rates of a Hydra service, or on an extreme criterion assigned to a QoS property.

One of the main Lessons Learned from the second year prototype was that in focus of networking embedded systems, especially QoS semantic service selection is required to support the implementation of intelligent ambient applications. There are two possible ways to do service selection: the average calculation of fitting rates, i.e. how well a QoS property matches to support a given task, or the definition of an extreme criterion for a QoS property.

Moreover, to apply during semantic service selection (mathematic) loading, i.e. relating different weights for a set of QoS properties is an adequate way for selecting the best-suitable service. For instance, property A may count twice than property B for decision-making. For future work, this represents an interesting criterion in regard of semantic service selection.

Furthermore, it is quite helpful to provide mechanisms for monitoring QoS properties—static and dynamic QoS properties as well—for changes, and forward these changed values to self-management component for QoS-based self adaptation, configuration, and self-protection.

A further noticeable Lesson Learned from the second year prototype was to consider further variables besides QoS properties for selecting the best-suitable service. In particular, we found that in addition to QoS we need to consider the context. Such a combination of QoS and context awareness may improve semantic service selection, though both kinds of parameters may overlap, depending on how things are viewed.

Not least, this written report provides a real-world scenario that shows QoS integration of Hydra middleware. This example is described in detail in the chapter 5. Basically, it demonstrates how to use QoS enabled Hydra middleware for applying semantic service selection in a pervasive environment for the home automation domain.

1 Introduction

1.1 Purpose, context and scope of this deliverable

D4.10 Quality-of-Service Enabled HYDRA Middleware is a prototype deliverable that explains its concept and the content of the provided software.

The **Q**uality-**o**f-**S**ervice (QoS) Enabled HYDRA Middleware is the result of the work of WP4 within Hydra during the third iteration of the project. However, we have noticed during this iteration that this work is strongly related to the application specific work that has been accomplished and is still ongoing in WP6. The starting points of this deliverable are D4.5 [3], D4.7 [4], and D8.3 [5], and not least lessons learned from the second year prototype.

According to the project time plan, the QoS related middleware components will be integrated in this month (M36) by WP4 in order to provide complete Quality-of-Service enabled Hydra middleware distribution.

This report provides an update on the description and implementation of the QoS related tools and middleware managers, which WP4 is responsible for:

- Ontology Manager, and
- QoS Manager, and
- Self-Management Component.

The managers and tools have evolved in order to reflect the lessons learned from the second year prototype. Moreover, interdependencies and synergies among managers and tools have been clarified in order to avoid redundant development and to draw a line of the specific functional scope for each component. In particular, the main focus of this deliverable is on the QoS Manager, though we try to make clear the relationship to Ontology Manager and self-management component.

Regarding the DDK, this report provides a set of guidelines for device developers for defining QoS properties for the set of services running on heterogeneous embedded devices.

The intended audience of this report are both:

- Application developers that are primarily specialized in the field of ambient intelligent networked embedded systems, and who plan to use Hydra QoS enabled middleware facilitating them for building a wide variety of applications, and
- Device developers that need to keep on track of changing QoS property data values resulting from monitoring regularly QoS properties in Hydra ontology.

1.2 Background

The reader of this document is supposed to have general background knowledge of SOAP Web Service concepts, such as XML, WSDL, SOAP and UDDI. Further, it is expected that the reader has basic knowledge of Quality of Web Services, semantic web, ontology and other related concepts. Referring to Hydra middleware, we strongly recommend to have a general overview of the Hydra Middleware architecture and its components explained in detail in D3.9 [6]. Referring to possible sets of QoS properties D4.8 [7] provides information on relevant parameters for Hydra middleware and applications built on Hydra. Moreover, in respect to a later on described QoS real-world home automation scenario, we suggest to have basic knowledge of Hydra Network Manager elaborated in D5.10 [8].

2 Quality of Service

2.1 Summary

This chapter focuses on the general meaning of Quality of Service (QoS). It shortly explains where the term QoS has its origin, and how we deal with it terms of Hydra middleware. Then it leads into QoS for computer networks, and builds a bridge to QoS for Web Services and what general architecture is required to support this. Finally, this chapter fades into the related work on QoS semantic Web Service selection.

2.2 General

Originally the term of QoS emerged from the computer networking area, in particular from the packet-switched telecommunication networks area. In computer networking and other packet-switched networks, efforts were made in the switching protocols to provide qualities such as bandwidth, jitter, and packet loss in addition to best effort service. And in respect of Hydra, such effort is shift from the network level to the application level. The QoS we are focusing on inside the project is called semantic QoS or QoS for Web Services. Hereby, we strongly focus on semantic service selection for a similar set of services which are annotated with quality views of service properties (see chapter 3 for detailed explanations).

2.3 QoS for Networks

Interconnected set of networks that are joined by routers are called packet-switched networks. The Internet is a global system that consists of interconnected, large packet-switched computer networks. Without applying QoS, every service delivery in the network is in its best effort. That means obtaining unspecified variable bit rate and delivery time, depending on the current traffic load, there is no guarantee that data is delivered. For example, bit rate, delay, jitter, packet loss cannot be qualified. The network capacity is strongly utilized; technologies as real-time streaming multimedia applications increase on the Internet. Therefore, it is realised that quality of service guarantees are important.

For usual IP based networks, efforts which enable QoS for data delivery in the network layer can be mainly divided into two approaches. There are Integrated Service (IntServ) described in [9] and Differentiated Service (DiffServ) described in [10].

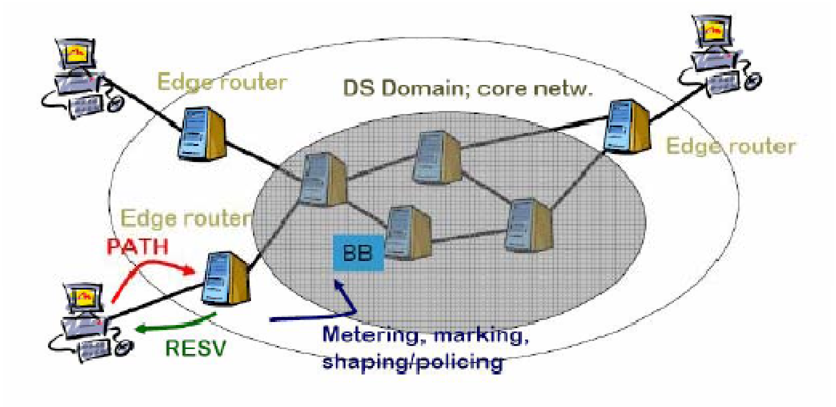


Figure 1 DiffServ Schema

- IntServ, is an extension of the IP architecture for providing QoS guarantees in networks with flow-based mechanism. It relies on resource reservation, and routers need to maintain the state information of allocated resources and respond to new call set up requests. The four main components are:

   - Packet Scheduler,
   - Classifier,
   - Admission Control Routine, and
   - Reservation Set-Up Protocol (RSVP).
- DiffServ, architecture improved from the fine-grained flow based mechanism of IntServ. It classifies flows into classes to providing appropriate QoS. There are two major components added: Packet Marking and Per Hop Behaviours (PHBs).

	IntServ	**DiffServ**
Service Differentiation	End-to-end	Local
Scope of Service Differentiation	A Unicast or Multicast	Anywhere in a network or in specific paths
Scalability	Flow-based, limited by the number of flows	Class-based, the number of classes of service
Network Management	Similar to Circuit Switching Network	Similar to existing IP networks

Table 1 Comparison between IntServ and DiffServ

Furthermore, the efforts can be supported by other types of packet-switched networks, such as networks with Multi Protocol Label Switching data carrying mechanism.

2.4 QoS for Web Services

2.4.1 General

Web Services are a middleware technology that facilitates the development of distributed applications that can be discovered, described and accessed based on XML and web protocols over intranets, extranets and Internet [11]. In the service-oriented architecture (SOA) paradigm it is simply called “services”. As mentioned in the previous section, the original QoS works as the fingerprint to guarantee data delivery on the network layer. However, in the Web Service consuming environment, for example the Pervasive Computing [12] environment as Hydra middleware targets, there is similar necessities for guarantee in the application level. Quality parameters such as performance related parameters (e.g. availability, response time) and service consumption related parameters (e.g. cost) vary between different distributed applications. The necessary negotiation about all those non-functional quality related service properties improves the satisfaction of the service requester—a user or another service—during service consumption. Thus, regardless of how it emerged, the intention and also the effect of supporting QoS enabled Web Service selection is not as same as they are in original packet-switched communication area.

The standard major requirements for supporting QoS in Web Services are as follows [13, 14]:

- *Availability* is the quality aspect of whether the Web Service is present or ready for immediate use. Availability represents the probability that a service is available. Larger values represent that a service is mostly ready to use while smaller values indicate unpredictability of whether the service will be available at a particular time. Moreover, availability refers to time-to-repair (TTR). TTR represents the time it takes to repair a service that has failed; ideally smaller values of TTR are desirable.
- *Accessibility* is the quality aspect of a service that represents the degree it is capable of serving a Web Service request. It may be expressed as a probability measure denoting the success rate or chance of a successful service instantiation at a point in time. There may be situations a Web Service is available, nevertheless it is inaccessible. High accessibility can be achieved by building highly scalable systems. Scalability refers to the ability to consistently serve the requests despite variations in the volume of requests.

- *Integrity* is the quality aspect of how the Web Service maintains the correctness of the interaction in respect to the source. Proper execution of Web Service transactions will provide the correctness of interaction. A transaction refers to a sequence of activities to be treated as a single unit of work. All the activities have to be completed to make the transaction successful. When a transaction does not complete, all the changes made are rolled back.
- *Performance* is the quality aspect of a Web Service, which is measured in terms of throughput and latency. Higher throughput and lower latency values represent good performing Web Service. Throughput represents the number of service requests that is served at a given time period. Latency is the round-trip time between sending a request and receiving the response.
- *Reliability* is the quality aspect of a Web Service that represents the degree of being capable of maintaining the service and service quality. For instance, the number of failures per month or year represents a measure of reliability. In another sense, reliability refers to the assured and ordered delivery for messages being sent and received by service requestors and service providers as well.
- *Regulatory* is the quality aspect of the Web Service in conformance with the rules, the law, the compliance with standards, and the established Service Level Agreement (SLA). Web services use a lot of standard technologies, such as SOAP, UDDI, and WSDL. Strict adherence to correct versions of standards (for example, SOAP version 1.2) by service providers is necessary for proper invocation of Web Services by service requestors.
- *Security* is the quality aspect of Web Services providing confidentiality and non-repudiation while authenticating the parties involved, encrypting messages, and providing access control. Web service invocations happen over the Web, and thus secure-based communication represents an important feature. The service provider can have different approaches and levels of providing security in order to satisfy the diverse needs of the service requestor.

For Web Services in general this requirement list covers a lot of aspects. However, in terms of Hydra we believe that *Efficiency* instead of *Performance* is more reasonable. *Efficiency* denotes the performance relative to the resources consumed [15]—e.g. two services which both has response times of 10ms may perform equally in absolute terms, but if one of them requires twice the amount of memory and CPU cycles to maintain this level of performance, it is less efficient. So, if you consider a set of quality attributes relevant to QoS in Hydra, *Efficiency* is the better choice, because it goes beyond *Performance,* as it additionally considers the energy awareness playing an important role for services being deployed on resource-constrained, embedded devices.

2.4.2 QoS Enabled Architecture

In general, in a three-tier Web Service architecture (see Figure 2), a syntactic-based description language, like WSDL for SOAP Web Services, is used to describe the signature for a service during service advertisement. After registering the information described by WSDL, the service registry such as UDDI will locate the target services according to service requester's functional requirements (e.g. some key words). However such a service invocation scheme does not assure the knowledge of any expected quality of service issues. The reason for this is that WSDL syntactically lacks semantics on non-functional aspects and the discovery model in current UDDI only supports searching by categories.

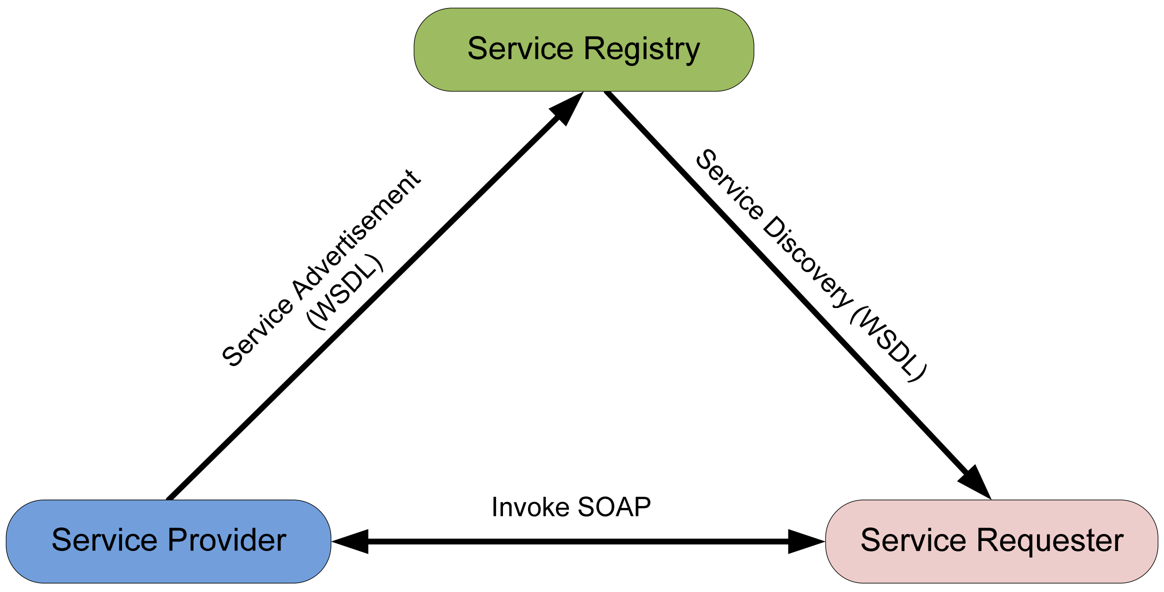


Figure 2 Three-tier Web Service Architecture

In order to 'QoS-enable' the Web Service Architecture, three kinds of efforts are required:

- First, to enrich the semantics in the description language, it is necessary to make it capable to specify non-functional information. The Semantic Web technology is used, providing more powerful ontology instead of pure WSDL to describe the service. For instance, OWL [16] is one of the ontology language used to describe semantic Web Services.
- Second, to extend the discovery model in the service registry for capability-based searching, besides being capable for searching by key words. In [17] is explained how to extend the *tModel* to make the discovery model QoS enabled, and [18] describes the mapping algorithm in the relevant state of art applied application OWL-S/UDDI matchmaker (see Figure 3).

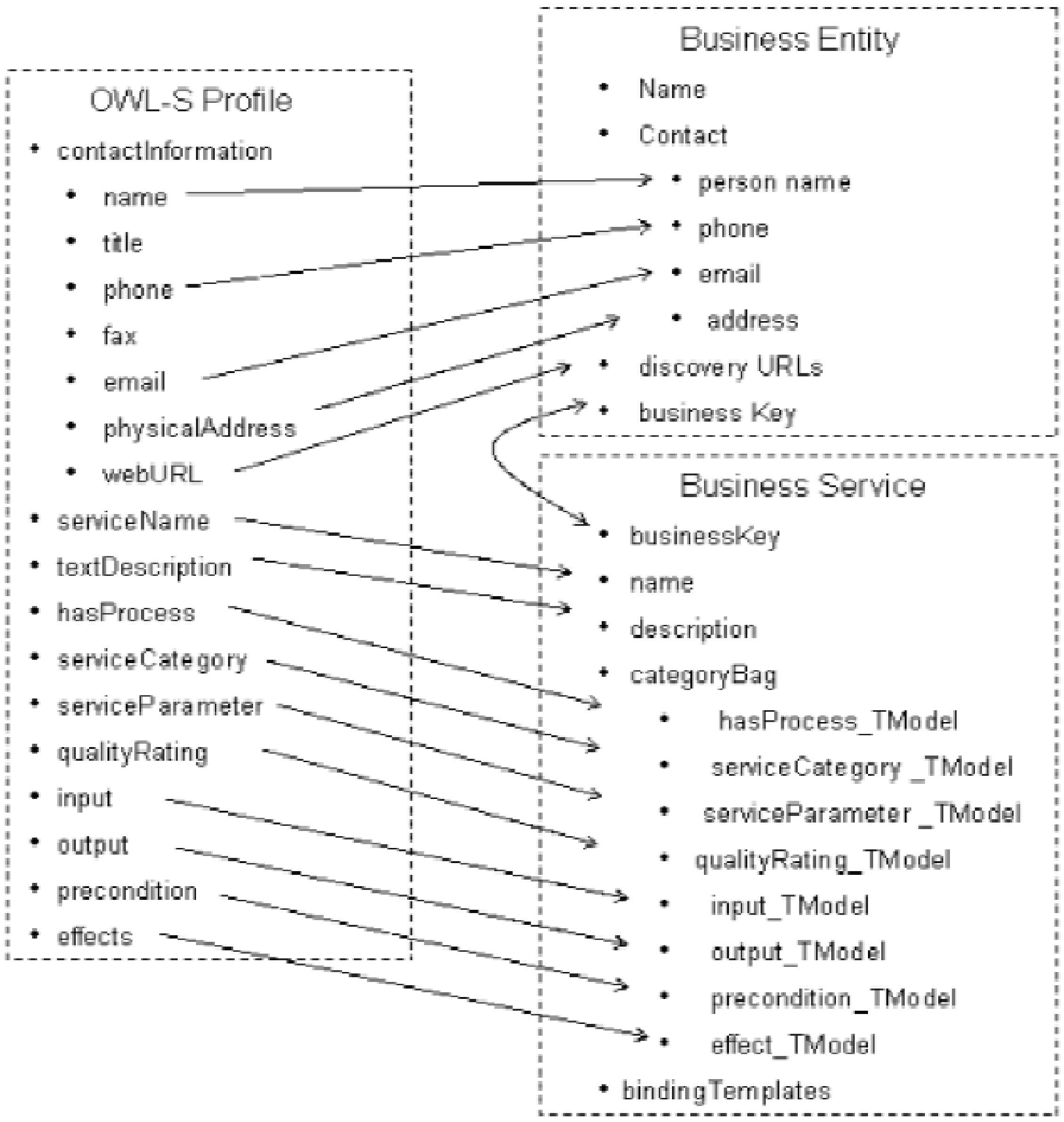


Figure 3 Mapping between OWL-S profile and UDDI [18]

- Third, instead of three-tier Web Service architecture, broker based Web Service architecture is used, i.e. during advertisement and service selection phase a broker is acting between the service consumer and the service provider (see Figure 4). Inside the broker the multi-criterion reasoning is executed for semantic service selection [19].

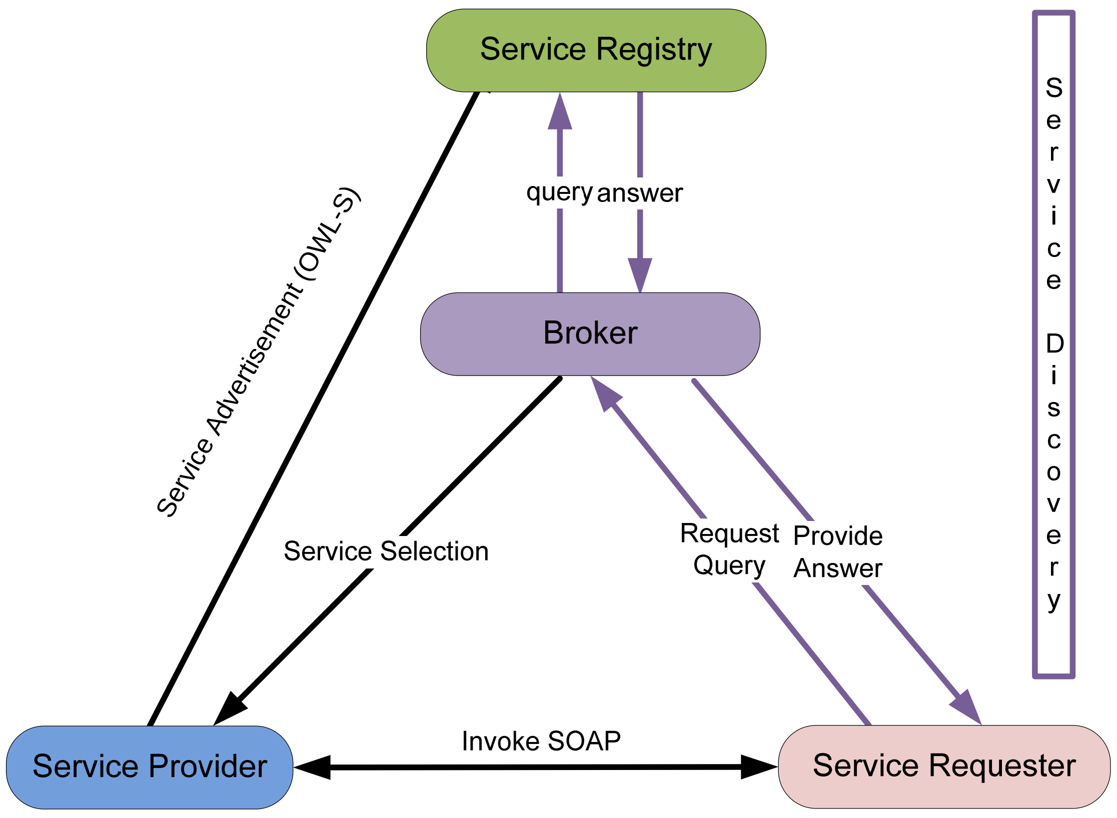


Figure 4 Semantic-based Web Service Architecture

2.4.3 Related Work

Some acknowledged research works focus on semantic Web Service selection; in particular QoS enabled Web Service selection. However, a lot of this resides at the theoretical or laboratorial stage—they analyze its necessities and also provide some related frameworks to support QoS based service selection.

We think there are two angles of view how to deal with the service selection problem. Hereby, the main difference depends on whether the service selection is processed by a single service or by combined services building a service chain, i.e. in former case an atomic service, and in the latter case a set of complex services [20, 21], is used. In the following both views are described:

- From an *end-user* point of view, regardless if services are atomic or complex, they are viewed as a whole. Just as a black box that has an inner, hidden functionality. Users mainly care what benefits a service provides. Hence, it makes sense to go beyond service selection that considers merely functional issues. Thus, to do service selection while considering quality properties criterion, i.e. non-functional properties are valuable. Research work reporting on OWL-Q [22], WSMO-QoS [23] and QoS Ontology Language [24] all state the similar daily life service consumption problem, that when selecting services only by functional properties, i.e. through key-words searching mechanism, a large numbers of similar services will return. Each work provides its designed QoS ontology model for service description and proposes the accompanying matching algorithm to solve the problem. Both OWL-Q and WSMO-QoS provide modular approaches with flexibility of extending, though they ignore any concrete definition of common QoS properties, and thus having week support for using their set of ontology. The QoS Ontology Language separates the ontology into a vocabulary and a language level. It almost collects a complicated list of common QoS properties in its vocabulary and its powerful metrics as well as relationship design enable representing any arbitrary QoS attributes. However, its shortcomings are its weak support for the actor (e.g. service provider or requester) and its overhead functions for reuse (e.g. conversion).

- From the *system* point of view, especially in terms of Hydra we focus on heterogeneous networked embedded systems, each component in a large system environment is kind of an inner service. Some complex functions are composed by several inner services; services are inside the same environment. Services are composed in a static and dynamic way as well. Compared to the static composition, a dynamic composition requires additionally user's request, context and preferences information—in this case the user may be represented by an end-user or some high level service within the service chain. [25, 26] describe a semantic service selection component as a part of their whole dynamic service composition framework, though they focus on context information, and neglect QoS parameters.

As Hydra is a pervasive middleware for networked heterogeneous embedded devices and services, it consequently needs to deal with the service selection problem in order to support service requesters to select the best-suitable service out of a range of similar services. For the reason that QoS is only one aspect of Hydra's self-management component [7], we have created our own set of ontology including the QoS facets in the Ontology Manager. For supporting better ontology usage, besides enabling arbitrary quality attributes, our design includes some concepts of common quality related properties, which are stated in section 3.3.1. There is a specific QoS Manager that works between the Ontology Manager and self-management component in order to bridge the two angles aforementioned. On the one hand, it offers negotiating result to the service requester after multi-criterion computation and decision-making, and on the other hand it collects the quality specific information that the self-management components requests regularly for QoS based self-adaptation inside the system (see section 3.4.1).

3 Semantic Service Selection

3.1 Summary

This chapter introduces QoS enabled Hydra Middleware. Further on, it explains the related concepts for defining QoS properties in Hydra Ontology. Afterwards, it explains in detail the concept of Hydra QoS enabled semantic service selection, and not least elaborates the functionalities of the Hydra QoS Manager as a middleware component.

3.2 QoS in Hydra Middleware

In 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 importance of QoS-enabling Hydra Middleware and the specific requirements definitions for accomplishing this issue is dealt with in D2.5 [27]. The set of QoS properties Hydra deals with depends on the specific application domain. Though, one learned lesson of the third iteration was that "cost" as a property is very important for service requestors. On the other hand we found out that 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.

So, conclusively, in Hydra the considered QoS service properties are listed below. They are categorized for different related quality "views of services":

- Service Level Performance Related View
  - Availability: The percentage of time when the device is connected and bonded service is available.
  - Throughput: The amount of data transfer ability.
  - Response Time: The response speeds. The time is estimated with perceived delay.
- Service Consumption Related View
  - Cost
  - Power Consumption Efficiency
- Service Function Related View
  - Security
  - Error Rate
  - Reliability
  - Accuracy
  - Etc. (depends on each specific application domain)

Three properties in *Service Level Performance Related View* display the network related performance. They are similar as Network Performance Metrics (NPMs) for network level performance properties with the same purpose, but the data for those service level properties are acquired using another way. Compared to network monitoring technology used to measure NPMs, here; some third party software or methods are used to test the parameters from the application level. And, the properties in this category will be used as performance criterion for semantic service selection.

Two properties in *Service Consumption Related View* display the important information for using the service. Both of them work as necessary criterion for service selection.

Three properties in *Service Function Related View* as *Security, Error Rate* and *Reliability* are mandatory in each domain and others as *Accuracy* and other unlisted ones depend on different application domains. For instance, applications in the e-health domain may have accuracy as service property. These properties will be considered as functional quality criterion for semantic service selection.

Furthermore, all the properties will be forwarded to self-management component for QoS-based self-adaptation, configuration, and self-protection. D8.4 [28] section 3.8.1 elaborates this issue.

3.3 QoS enabled Hydra Service Selection Mechanism

3.3.1 Defining Concepts in Ontology

QoS refers to a service level satisfying users that might not necessarily be the best service level, though the one meeting intended to meet the user requirements. The user requirements are described afterwards by Service Level Agreement (SLA). 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. At the present stage of this deliverable as per the requirements for self-management, are summarized in the following list that contains the QoS parameters that should be considered:

- Bandwidth, or throughput
- Latency,
- Error Rate,
- Availability, including both network availability, and service availability,
- Reliability
- Accuracy (of measurement, operation),
- Speed (of operation, service),
- Power Drain (of service execution, of operation), and
- Cost

QoS ontology—actually it is 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 [29] QoS ontology and the OWL-Q [30] ontology. It simplifies the OWL-Q ontology in which we adopted its QoS specification idea, and included our listed parameters.

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 [7]. The part of QoSMetrics ontology is shown below in Figure 5.

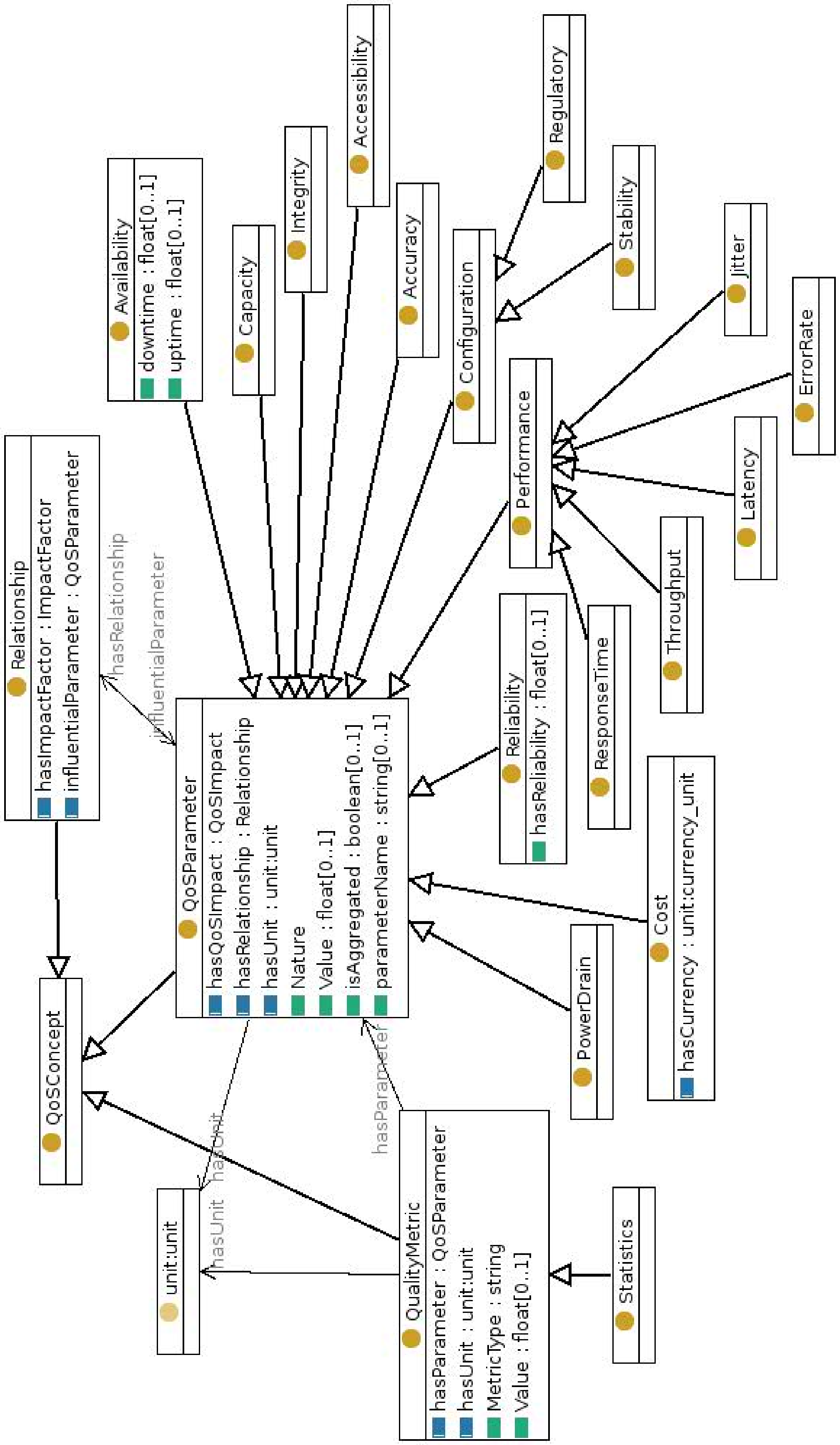


Figure 5 Ontology Parts for QoS Metrics

Another basic QoS supporting ontology is the Unit ontology that covers the wide set of various basic and derived units (e.g. the *second* is the basic unit, the *hour*, *day* or *month* are derived units). Figure 6 depicts the parts of Unit Ontology.

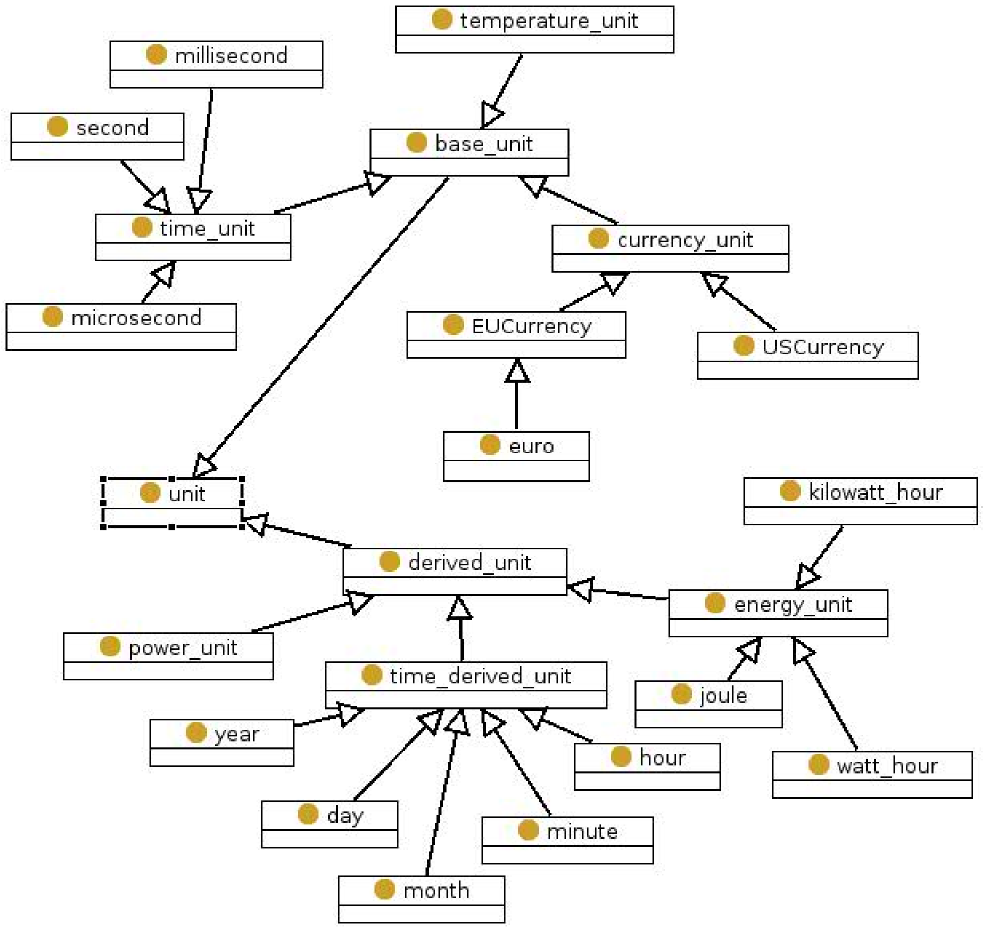


Figure 6 Part of Unit Ontology

Some of the QoS aspects, such as security or energy profiles, can be modelled separately by an ontology set, which precisely describes the specific concepts and properties of particular domains. These aspects can be annotated to the devices or services directly, or can be imported as the part of the QoS ontology.

The QoS ontology based on the QoSMetrics and units models defines the qualities of services in the terms from network, communication, or transport domain. When analysing the QoS properties of services, there can be also another interpretation of qualities specification depending on the purpose of qualities usage. In some cases, the requested quality of services may be formulated in the terms of required functionalities or other specific properties of desired services. For example, list all services, which are capable of playing the video files in the AVI (Audio Video Interleave) format. In this case, the functionality of services and the properties of service parameters can be used as the search criteria. The desired functionality of the services is modelled as the concept hierarchy of services. Additional functionality description can be annotated to services using the instances *FunctionalCapability* concept, which can be reused for more services if, needed. The service input and output parameters can be for this purpose annotated to the instances of unit ontology. The same way, the energy consumption or several security aspects of services can be used as the suitable service search criteria. An example of the part of functional QoS model is illustrated in Figure 7.

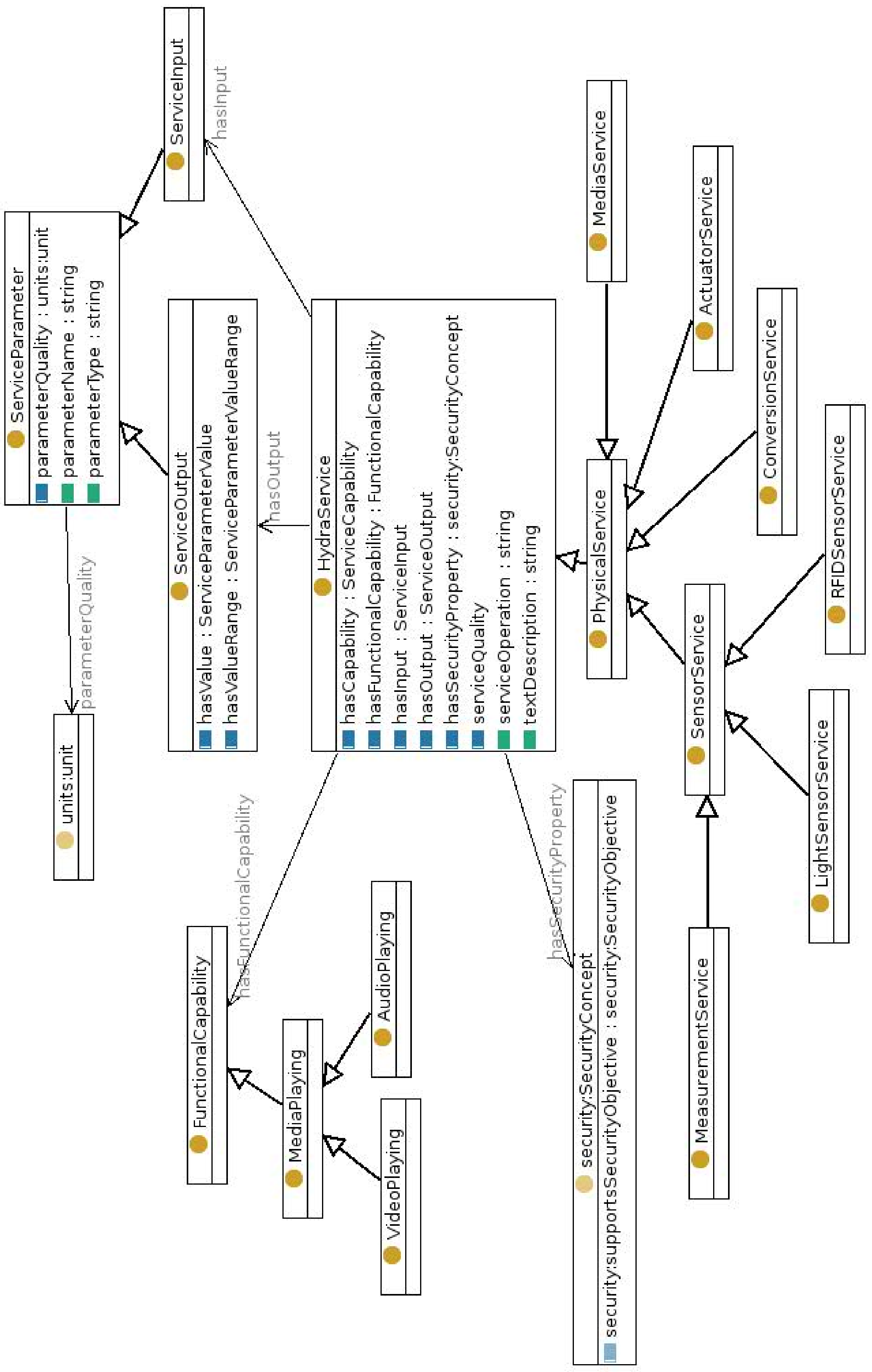


Figure 7 Part of functional QoS Interpretation Model

The basic *HydraService* class is sub classed to the service hierarchy. *ServiceParameter*s are annotated to the QoS *Unit* concept. A *HydraService* itself refers to the reusable *FunctionalCapability* hierarchy and *SecurityConcept*s. Furthermore, the main HydraDevice concept (not in the picture) refers to the *EnergyProfile* ontology, which specifies the energy consumption of particular devices. Using these two interpretations, the quality of service may be specified in the both ways:

1. Defining the service requirements in the terms of network or communication properties, and
2. The functional, security or energy consumption of particular services or devices providing them respectively.

3.3.2 Quantification of Service Matching Requirements

On purpose that the service resource consumption inside Hydra can be guaranteed with sets of important quality properties, specific application level quality parameters will be queried for multi-criterion based computation, i.e. it is quantified how well a service matches a requirement. The outcome result will be used as the basis for later decision making.

Though applications vary in their functions, they are domain based. Each domain associates with a set of specific important quality properties, which worth negotiation. For instance, applications in the home automation domain require the criterion of online transform speed, perceived latency for data stream, and not least the price. However, applications in the e-Health domain require the criterion of accuracy and reliability. Different from the network level quality guarantees (mentioned above in section 2.3), which need real-time monitoring and compared to available network resource, application level QoS is relatively static. This quality related information originally described from service description and acquired by system ontology manger and then specially queried for negotiation. Possible change only happens when the service description changes and any change will be casually or frequent notified. So, with rather static parameters, the corresponding computation could be static, methods used to treat static elements fit the need.

Request Oriented Percentage Ranking we call the method we use in Hydra for computing how a QoS property matches another. After querying all the domain specific quality properties for each available candidate services, according to the request from the user or composite service in the service chain, the standard is defined.

There are `least`, `most` and `sameAs+value` standards. They are extreme standards. For example, if the request is for getting the 'cheapest price', then the cheapest service will be marked as 100% fitted, and the most expensive will be 0% fitted. Other services will be percentage ranked accordingly. If the extreme standard is `sameAs+value`, only the service with exact value for such particular property is 100%, others are all 0%, as they are different.

There are other standards as `less` and `more`, which are similar as `least` and `most` during ranking time. They just require normal order from large to small or from small to large. The only difference between them and `least` and `most` is during decision-making. `less` and `more` do not belong to an extreme rule.

There are standards as `lessThan` and `moreThan` in order to satisfy more special request such as price less than 2 euro. For example, it will make every services with cost property less than 2 euro 100% fitted and the most expansive one 0% fitted, then the relative similarity will be computed for the services cost between 2 euro and the cost value of the most expensive service. The similarity will be converted to fitting percentage.

Therefore, seven standards work for computation: `least`, `most`, `sameAs+value`; `less`, `more`; `lessThan` and `moreThan`.

3.3.3 Decision Making

As described in D3.9 section 7.4.2. The decision making for QoS-enabed service selection in Hydra supports both standard and extreme request. According to the information from the rule engine, if the client requester defines "`least`, `most`, and `sameAs+value` standard request, that means it request the service fulfils the extreme rule on particular property and such request will with the highest priority in decision making algorithm. For example, suppose there are five candidate services providing the same functionality:

Service	Availability (in %)	Cost (in €)
Service 1	99	3
Service 2	98.7	2.4
Service 3	99.5	2.7
Service 4	97.9	2.5
Service 5	99.4	3

Table 2 List of Service property values

And the client requests for `Availability` property is the `most` and for costs `lessThan` 2.5 €. After computation the ranking list for `Availability` property is:

Rank	Service	Fitting (in %)
1	Service 3	100
2	Service 5	94
3	Service 1	72.5
4	Service 2	56
5	Service 4	0

Table 3 Ranking for Availability property

And, for Cost property the ranking list will be:

Rank	Service	Fitting (in %)
1	Service 4	100
2	Service 2	100
3	Service 3	60
4	Service 5	0
5	Service 1	0

Table 4 Ranking for Cost property

Since, an extreme (availability most) property has the highest priority, Service 3 with 100% for the *Availability* property will be list as the best choice, then the last candidates will be list according to the average ranking results. Thus, the result after decision-making is:

Rank	Service
1	Service 3
2	Service 2
3	Service 4
4	Service 5
5	Service 1

Table 5 Result after Decision Making

3.4 Quality of Service Manager - Functional Description

3.4.1 General

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.

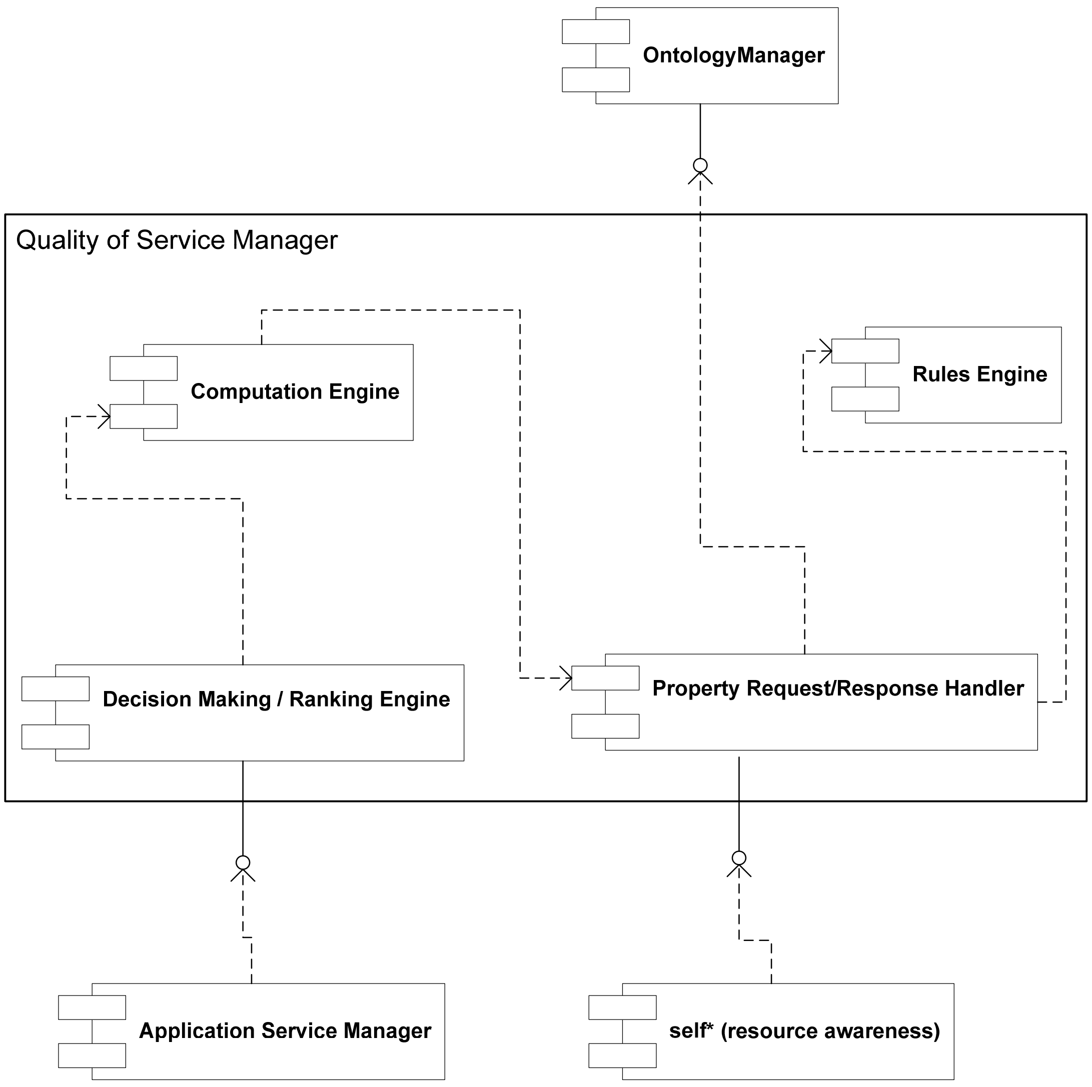

OntologyManager
Quality of Service Manager
Computation Engine
Rules Engine
Decision Making / Ranking Engine
Property Request/Response Handler
Application Service Manager
self* (resource awareness)

Figure 8 Software Architecture of Quality of Service Manager

These four internal sub-components are intended to be written in Java [31]. The goal is to keep each sub-component in a separate module to support the reusing, extending, or interchanging of specific modules. It is planned to deploy the modules as bundles that are conform to the OSGi specification [32], as it provides a dynamic module system for Java modules. The benefit for Hydra middleware of using the OSGi framework [33] upon other web containers as Tomcat [34], WebSphere [35] etc, is explained in D5.10 [8] section 3.2.2. Not least, it is planned to develop these bundles in the manner of OSGi Declarative Services specification [36], as it is more elegant way to deal with required dependencies in the form of XML than to use programmatically a ServiceTracker [33] object class per bundle.

In the following subchapters each sub-component will be elaborated.

3.4.2 Property Request/Response Handler

In Hydra middleware the QoS Manager depends on Ontology Manager which provides the **`getProperties(String [] queries)`** interface for querying the quality related properties. A query array composed by a SPAQRL String will be forwarded to the Hydra ontology model and return back the aiming data value sets. For example, if the query string of **`SELECT ?device ?power WHERE ?device hasPower ?power`** is forwarded, then the answer from the Ontology Manager will be sets of devices id and their power properties.

However, which possibilities of quality properties need to be queried, depends on the client-side requests, they may have been invoked from a person operating with his device or from some composite service in a service-chain. Inside Hydra services are deployed on a range of devices. Therefore, the services are somehow device-based, and an interface between the property request component and Device Device Manager is needed. For this the request component offers a web interface, as **`String [] getRequest (String deviceId)`**. The below figure depicts the process of querying and requesting QoS properties.

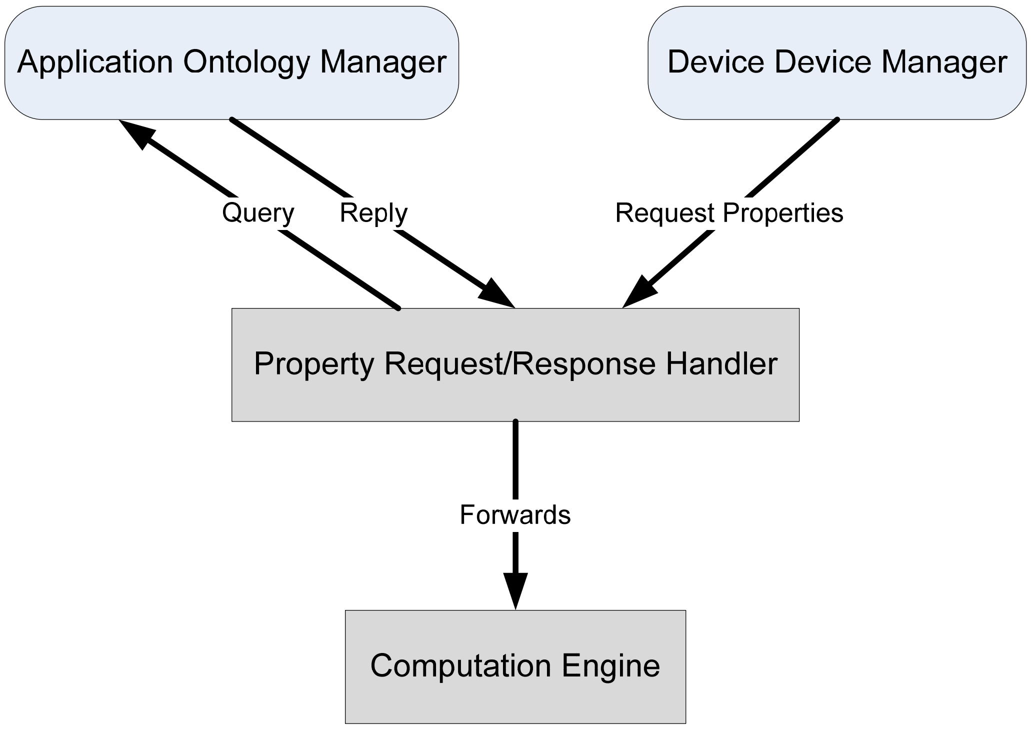


Figure 9 Overview of Query and Request Invocations

3.4.3 Rules Engine

The rule engine is intended to provide an interface to access the working device's request on consuming the service resource. Besides its requested non-functional property's name or ID, seven kinds of request standards can be categorized as: `least`, `most`, `sameAs+value`; `less`, `more`; `lessThan`, and `moreThan`.

As explained above (in section 3.3.2), after receiving from Ontology Manager a specific set of QoS properties, the computation engine calculates the matching of a property value to another property value. The computation on each quality property will use the *Request Oriented Percentage Ranking* method, and exactly the standards for performing the ranking the rule engine does provide (see Figure 10).

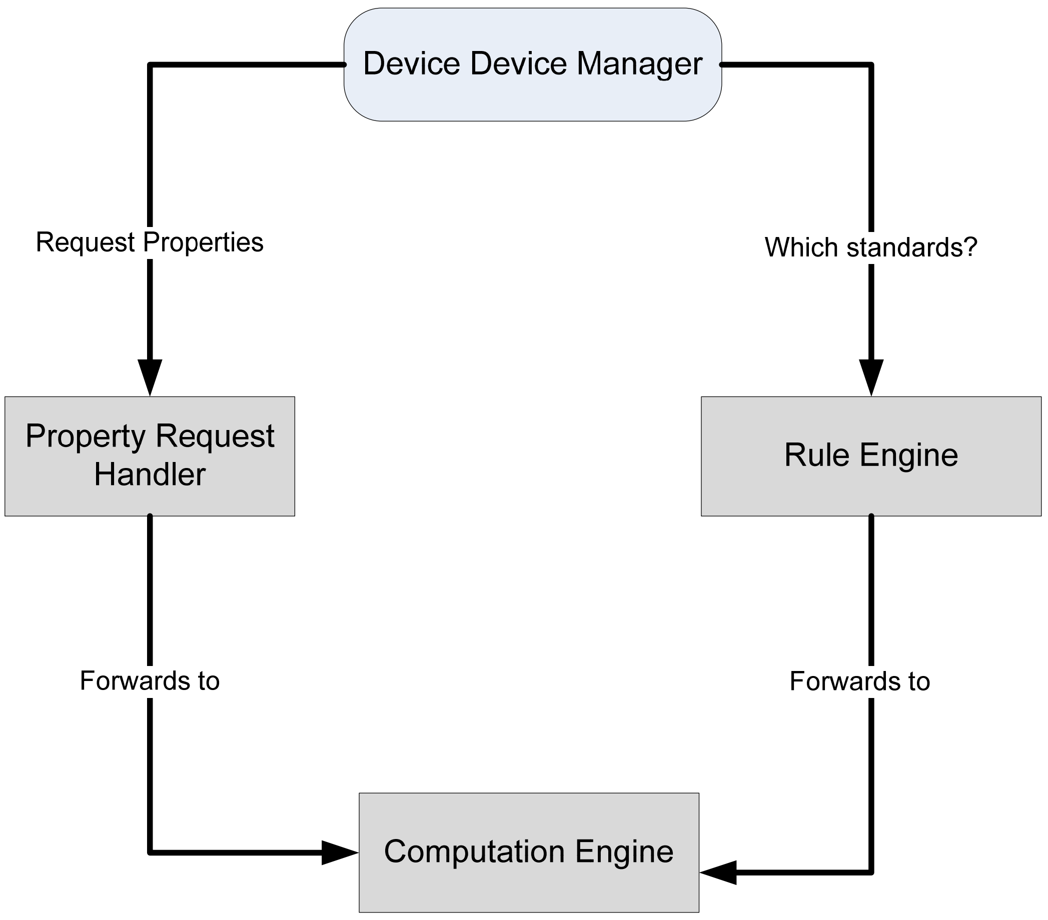


Figure 10 Interdependencies between Device Device Manager and QoS Manager

3.4.4 Computation Engine

The fitting percentage on each request property is computation-based on the standards from the rule engine. If the standard is some extreme restriction, the ranking will be ordered normally from most to least or from least to most and then convert to fitting percentage. If the standard is **`sameAs`** with some specific value, the similarity will be computed and convert to fitting percentage. If the standard is `less` and `more`, the computation is also the same. If the standard is `lessThan` or `moreThan` with some specific value, all the services on the trend are fulfilled as 100% and the similarity will be computed for others and the similarity will convert to fitting percentage. The computation in itself is rather static than dynamic. That is due to the specifics for each application domain you need to take into account.

3.4.5 Decision Making Engine

Getting the computation data from the computation engine, i.e. the fitting average rates describing how a service property matches a requirement, subsequently the decision for selecting the best-suitable service will be made. If the rule is standard, i.e. as default, the decision will be made by just averaging the fitting percentages. If the rule applies extremes, then the particular quality property will be considered with highest priority and is put at the first place.

4 QoS Based Self Adaption

4.1 Summary

This chapter introduces into the importance on keeping track of changing QoS properties, which according property data values are forwarded to Hydra self-management component for QoS based self-adaption.

4.2 Observe QoS properties for Triggering Events

Self-management depends on being able to reason on the state of a Hydra network. For example, being able to deduce, whether processes have failed or stopped responding, or if the performance of the network communication is adequate. Therefore, the network status is monitored. To report regularly its related QoS properties is important.

Figure 11 shows which protocols are relevant for the development of Hydra middleware. As Hydra is using SOAP for Web Service calls, in particular interesting protocols are HTTP, SOAP, TCP, UDP and IP. To accomplish this, we instrument both the service and its clients.

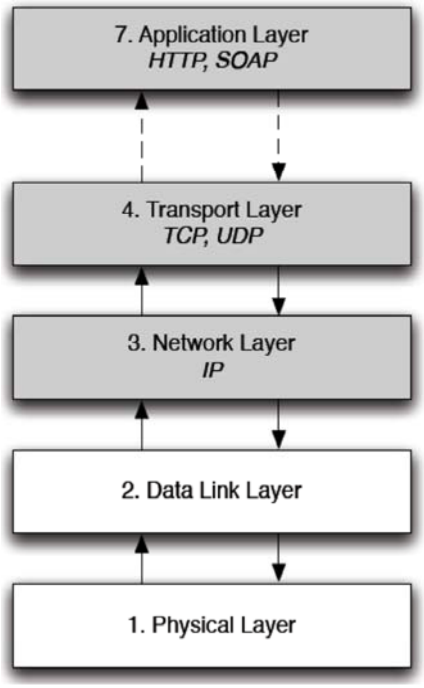


Figure 11: Overview of OSI Layers

4.2.1 Instrumenting the Service Host and its Clients

Figure 12 shows the intended deployment of our instrumental for service hosts called *Flamenco Probe*. It is intended to be deployed on all hosts from which network events may be reported.

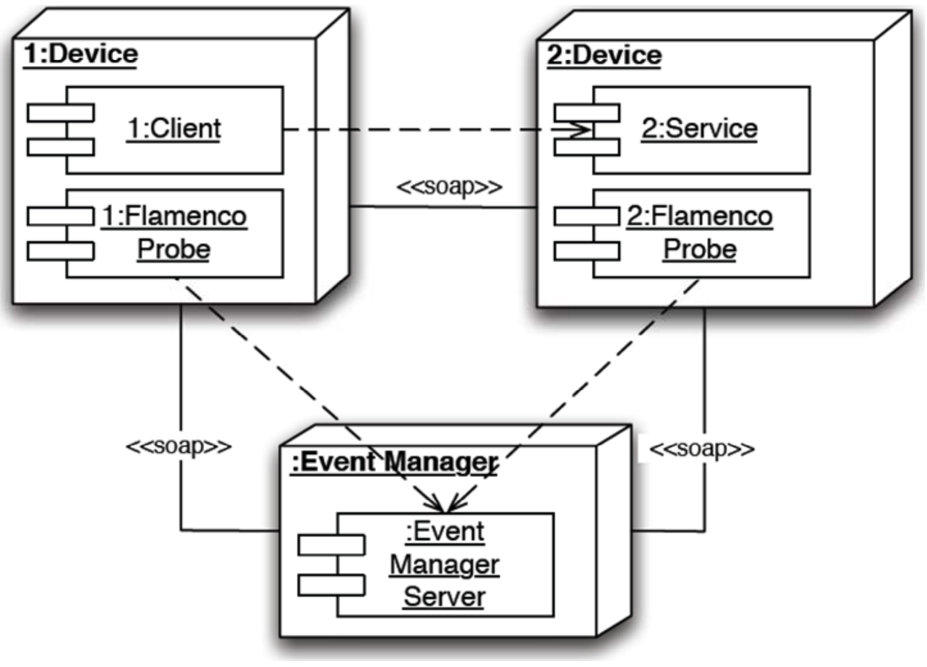


Figure 12 Flamenco Probe Deployment

Next, Figure 13 shows a dynamic view of the interaction between the Flamenco Probe and the Event Manager. The current implementation uses a Windows-specific version of tcpdump, which is named windump [37].

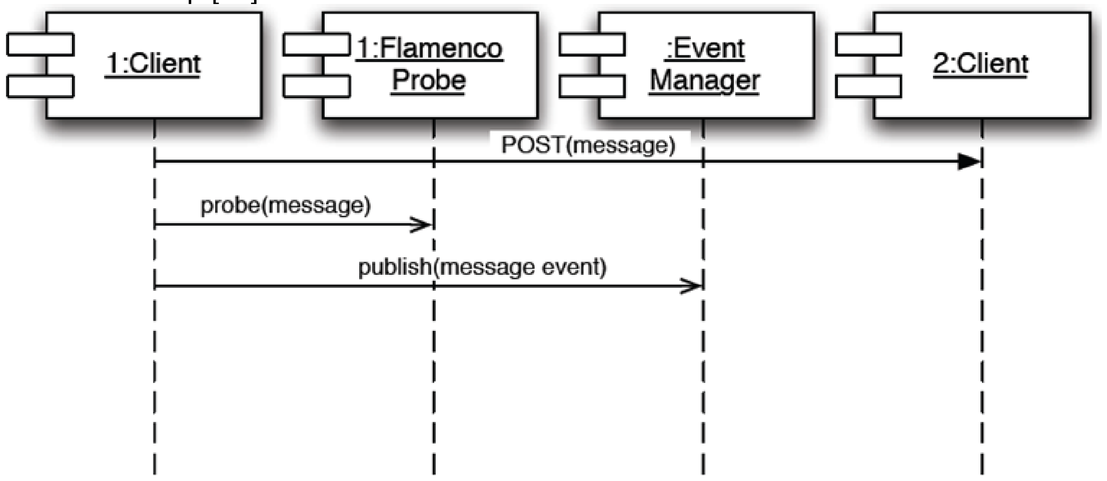


Figure 13 Flamenco Probe Dynamic View

4.2.2 MessageProbe ontology and calculation of QoS attributes

Accordingly, we have developed a MessageProbe ontology based on the aforementioned probe implementation. The details of this ontology are reported in D4.3 [38] and D4.8 [7]. Here is the rule for calculating the call of the round trip time (RTT), which represents one QoS metric. Further, it can be used to calculate the network latency, providing that the same time clock is used for both client and service, i.e. for both the time is synchronized:

```
ipsniffer:messageID(?message1, ?messageid) □
ipsniffer:messageID(?message2, ?messageid) □
ipsniffer:messageID(?message3, ?messageid) □
ipsniffer:messageID(?message4, ?messageid) □
```

```
abox:hasURI(?message1, ?u1) □
abox:hasURI(?message2, ?u2) □
abox:hasURI(?message3, ?u3) □
abox:hasURI(?message4, ?u4) □
swrlb:containsIgnoreCase(?u1, "clientbegin") □
swrlb:containsIgnoreCase(?u2, "servicebegin") □
swrlb:containsIgnoreCase(?u3, "serviceend") □
swrlb:containsIgnoreCase(?u4, "clientend") □
ipsniffer:messageSourceIP(?message1, ?ip1) □
ipsniffer:ipaddr(?ip1, ?ipa1) □
ipsniffer:ipaddr(?ip2, ?ipa2) □
ipsniffer:hasMessage(?process1, ?message1) □
ipsniffer:hasProcessID(?process1, ?pid1) □
ipsniffer:messageTargetIP(?message1, ?ip2) □
ipsniffer:initiatingTime(?message1, ?time1) □
ipsniffer:messageSourceIP(?message2, ?ip3) □
ipsniffer:messageTargetIP(?message2, ?ip4) □
ipsniffer:ipaddr(?ip3, ?ipa3) □
ipsniffer:ipaddr(?ip4, ?ipa4) □
ipsniffer:messageTargetPort(?message2, ?port2) □
ipsniffer:hasMessage(?process2, ?message2) □
ipsniffer:hasProcessID(?process2, ?pid2) □
ipsniffer:initiatingTime(?message2, ?time2) □
ipsniffer:messageSourceIP(?message3, ?ip5) □
ipsniffer:messageTargetIP(?message3, ?ip6) □
ipsniffer:ipaddr(?ip5, ?ipa5) □
ipsniffer:ipaddr(?ip6, ?ipa6) □
ipsniffer:messageTargetPort(?message3, ?port3) □
ipsniffer:hasMessage(?process3, ?message3) □
ipsniffer:hasProcessID(?process3, ?pid3) □
ipsniffer:initiatingTime(?message3, ?time3) □
ipsniffer:messageSourceIP(?message4, ?ip7) □
ipsniffer:messageTargetIP(?message4, ?ip8) □
ipsniffer:ipaddr(?ip7, ?ipa7) □
ipsniffer:ipaddr(?ip8, ?ipa8) □
ipsniffer:messageTargetPort(?message4, ?port4) □
ipsniffer:hasMessage(?process4, ?message4) □
ipsniffer:hasProcessID(?process4, ?pid4) □
ipsniffer:initiatingTime(?message4, ?time4) □
temporal:duration(?d1, ?time1, ?time4, temporal:Milliseconds) □
temporal:duration(?d2, ?time2, ?time3, temporal:Milliseconds)
  → ipsniffer:invoke(?message1, ?message2) □
sqwrl:select(?ip1, ?ipa1, ?pid1, ?ipa2, ?port2, ?pid2, ?d1, ?d2)
```

5 Future Work

5.1 Summary

This chapter deals with future possible work. The issue of assigning different loadings to QoS properties during Hydra QoS enabled semantic service selection is conceptually explained. This chapter closes with a discussion, that context related parameters in addition to QoS parameters need to be considered during the process of semantic service selection.

5.2 Loading of QoS Properties during SSS

Currently, Hydra QoS enabled middleware considers two kinds of decision making while processing semantic service selection (see also previous chapter):

- Standard Average Calculation: To compute for each QoS service property value the fitting rate (measured in percentage), and afterwards to calculate the average for the range of the whole fitting rates. The service with the highest fitting rate represents the best-suitable one.
- Extreme Calculation: To consider an extreme request. For example, a client requests to use only a Service that is running on a device that is grouped in energy efficiency class B. Therefore, the power consumption as a service QoS property would have an acceptable value in the range of 1-1.5 watt/hr. For example, we have two services: Service1 and Service2. If Service1 has a fitting average rate of 96% and is in energy efficiency class B, and Service2 has a fitting average rate of 84% and is in energy efficiency class A, then the decision making algorithm will select Service1. But, if Service1 would have been grouped into energy efficiency class C, the decision-making algorithm will select Service2, as it fulfils the extreme request.

During the second year prototype we have found out, that the function of decision making could be much more complicated. We think of assigning specific loadings among QoS service properties, i.e. each different criteria quality property may have different weighting during decision making. In the following we explain the loading by a little example:

Suppose there are three services (Service 1, Service 2 and Service 3) and three requested properties (power efficiency, cost and accuracy). For the *Cost* and the *Accuracy* properties they are requested through `less` and `more` standards. For the *Power Efficiency* property, it is requested through extreme standards `sameAs ClassB`. Thus, according to our designed computation algorithm, explained above in chapter 3, the ranking lists are as follows:

Ranked Services	Cost (`less` standard)	Percentage	Percentage (with weight 0.2)
Service 2	2.3 euro	100%	20%
Service 1	2.5 euro	49%	9.8%
Service 3	2.67 euro	0%	0%

Ranked Services	Accuracy (`more` standard)	Percentage	Percentage (with weight 0.8)
Service 3	99.99%	100%	80%
Service 2	99.9%	69%	55.2%
Service 1	99.7%	0%	0%

Ranked Services	Power Efficiency (`sameAs` extreme standard)	Percentage
Service 2	ClassB	100%
Service 3	ClassB	100%
Service 1	ClassA	0%

Table 6 Loading Properties Ranking Lists

Services	Average in Percentage	Average Percentage (with weight)
Service 1	24.5%	4.9%
Service 2	84.5% (*selected with extreme rule fulfilled*)	37.6%
Service 3	50%	40% (*selected with extreme rule fulfilled*)

Table 7 Fitting Rates in Percentage

The extreme rule has the higher priority than standard rules, but if we want to make difference among standard rules, the specific loading is needed.

There can be another "weight" value together with the request for displaying standards. The interface for request could change to `getRequest(Request request)`, and data object of Request is composed of `string: property, float weight and string:standard`.

For instance, with same example, now the requests are "Cost with standard `less` and weight 0.4" and "Accuracy with standard `more` and weight 0.6" Ranking after computation currently is multiply the weight factor (please see every last row for each table within this section).

5.3 QoS and Context Awareness

The development of Hydra middleware targets to support efficiently device and application developers with a comprehensible framework allowing them to focus on the specific issues of the application development without having to deal with the complexity of low-level issues in a pervasive environment, such as discovery of devices and services, the intelligent networking of heterogeneous systems, or security issues [1]. During the second year prototype we clarified that the context is a further variable we need to consider during the process of semantic service selection, besides the set available for QoS service properties. The term context has many interpretations, such as described in [39]. In Hydra we have defined in [40] section 3.1 context as: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." To consider the context in a system, i.e. to provide some kind of context awareness covers that the system behaves depending on the context a user is situated in. In particular, the system assists the user or takes into account the complex and subtle relationship between the environment and him, and thus tries to support more properly to accomplish his task [41].

Hydra middleware organizes all the services inside the pervasive environment device-based. In Hydra Ontology each description of a Hydra-enabled device—regardless of the device category—is combined with the Web Service description. In the face of *Dey's* definition for context *"Context is any information that can be used to characterize the situation of an entity."* [42], the Hydra ontology framework uses to some degree may involve the necessary context information [39] for some hardware. To enable the user to conveniently input personal profile information through devices, the ontology framework in Hydra is capable of accessing some static information about the user's situation, such as the user's gender, age and even his preferences if required.

However, as mentioned above, context is rather a broad term, there could be some knowledge model specially built for collection context properties. Though, Hydra middleware just makes use of some simple one. There is no particular ontology model built for a category dealing with context information. Concerning Hydra middleware the overlapping in the understanding for quality-of-service properties and context information has not been differentiated clearly yet. To stress the hazard of avoiding a clear separation between both kinds of variables, quality-of-service and context-awareness, we like to refer to the location problem described in D4.5 [3] section 4.2.1.

6 Application Scenario for QoS enabled Hydra Middleware

6.1 Summary

In this chapter we apply QoS enabled Hydra middleware for a real-world home automation scenario. In the beginning we tell the story, and depict a storyboard. Further, we view the scenario from a QoS enabled Hydra middleware angle, and present our dedicated mobile application. We apply the Hydra semantic service selection on the posing service selection problem, and finally conclude with results and future work.

6.2 Story

6.2.1 Story Telling

Lucy is doing her masters in computer science at the University of Hydra Technology. She is in the second semester, and is specialized in mobile, pervasive computing. That is why she has recently purchased a new cell phone equipped with some fancy functionality, such as projecting a video stream to another device capable of playing back a video. Now Lucy is walking through the streets on her way to her classes, and sees a poster advertising the new Pink album.

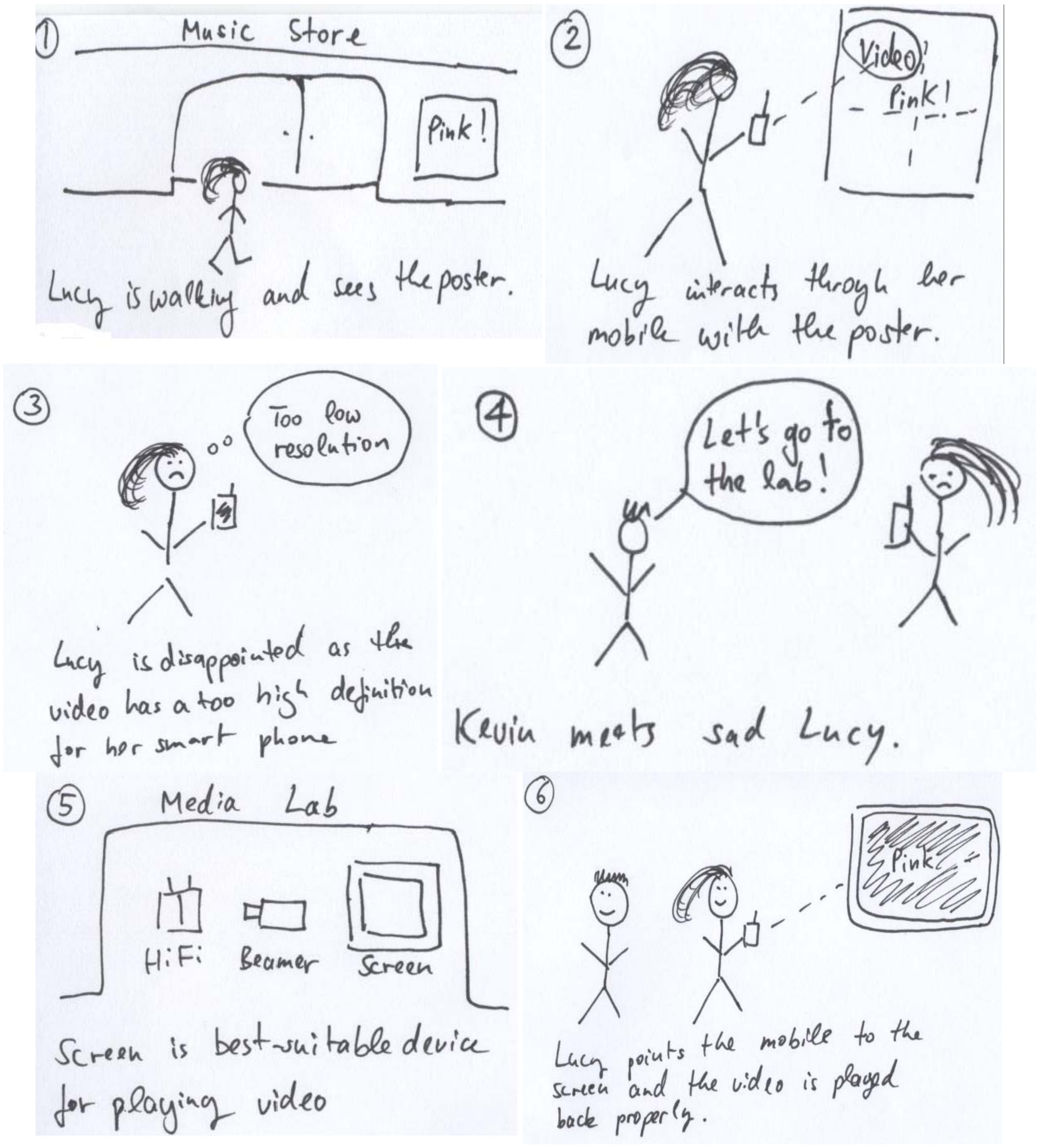
The poster consists of four action fields that Lucy can interact with while using her mobile device. First field provides the download of a video trailer, second for receiving an audiotape, third for connecting to the band's homepage, and last one directs you to the booking page for buying tickets. As Lucy likes to watch the video she scans the first action field.

Lucy downloads the video trailer and starts to play it on her mobile device. Unfortunately, she does not enjoy watching the video on her mobile. Especially the resolution of her cell phone is too low for playing properly the high definition video. Kevin, a PhD Student passes by and sees sad Lucy. As he knows Lucy is a big Pink fan, he suggests her to go to the home automation lab at the university, a place equipped with multimedia devices, and to watch there the trailer on a more appropriate playback device.

The home automation lab is equipped with a Hi-fi, LCD screen, and a beamer. Arriving at the lab Lucy and Kevin ask an application installed on Lucy's cell phone to scan the environment for 'PlayVideo' services. After some short while the mobile signals them that the LCD screen is the best-suitable device for playing the video. While carrying her mobile with an intuitive gesture 'take it' Lucy points at the LCD screen, and the trailer is played back on the LCD screen. Lucy enjoys watching the video, and invites Kevin for a coffee in return. In the evening both meet for a date, and they lived happily ever after.

6.2.2 Story Board

In the following a storyboard illustrates the real-world home automation scenario:


1
Music Store
Pink!
Lucy is walking and sees the poster.
2
Video
Pink!
Lucy interacts through her mobile with the poster.
3
Too low resolution
Lucy is disappointed as the video has a too high definition for her smart phone
4
Let's go to the lab!
Kevin meets sad Lucy.
5
Media Lab
HiFi
Beamer
Screen
Screen is best-suitable device for playing video
6
Pink
Lucy points the mobile to the screen and the video is played back properly.

6.3 Applying Hydra Semantic Service Selection

6.3.1 General

In the following we strongly focus on the scenario from QoS enabled Hydra semantic service selection view. In particular, we take as a basis the processing of the computation engine and decision algorithm explained above in chapter 3. In the described scenario, the service selection belongs to the *HomeAutomation* domain. For this domain four properties are defined:

1. *Resolution*,
2. *Screen size*,

3. *Power consumption*, and
4. *Colors*.

All four properties are set as default criterion for service selection. The user has the choice to assign one of them as extreme rule during service selection. In regard to our scenario, that means Lucy cares at most for the service running on a device that offers the highest resolution. Lucy has defined the Resolution property as the main criterion for semantic service selection, i.e. for the Resolution property the extreme rule is applied.

In the start Lucy can only consume the video her mobile phone, as no other devices capable of playing back the video are available in the environment. However, she is unsatisfied while watching the trailer on her cell phone, as the resolution of the phone is too small for the high definition video. Exactly, at this stage Hydra QoS enabled semantic service selection gets involved: Kevin and Lucy arrive at the home automation lab of Hydra University. There are several devices with the same *PlayVideo* function services available. After considering all of the four default properties for the *HomeAutomation* domain, and not least Lucy's choice for *Resolution* property as extreme rule the application returns back a list with the best three suitable services, indicating also the fitting percentages.

6.3.2 Set of Device-based PlayVideo Services

In the scenario three possible *PlayVideo* services are available in the environment to consume the video trailer. Each of the services is deployed on a specific device having particular values for hardware and software properties. For this simple scenario the *HomeAutomation* domain merely considers the power consumption, colors, screen size, and the resolution. Following table gives an overview for each device property data values:

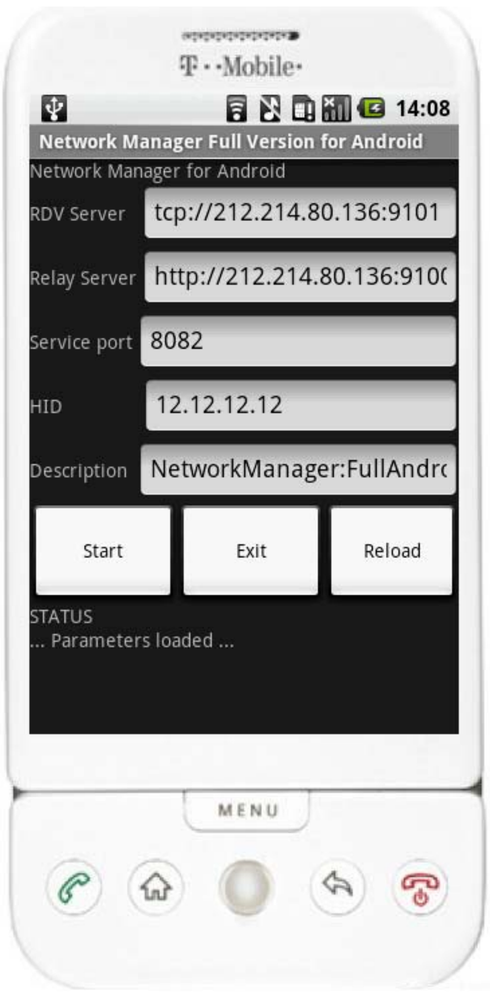
Device	Properties
LCD TV	Resolution: 1440*900 Screen Size: 19'' Power Consumption:56 watts Colours: 16.7 millions
Beamer	Resolution: 800*600 Screen Size: 50'' Power Consumption:48 watts Colours: 16.7 millions
Hi-fi	Resolution: null Screen Size: null Power Consumption:1 watts Colours: null

Table 8 Overview of Device HW/SW properties

For each device the various properties are registered through a specific device description. Then the Hydra Ontology Manager acquires the device properties with the units and the related services descriptions. During application run-time the specific property values are stored in a dedicated database.

6.3.3 User-Interface Interaction with Scenario-based Application

In the scenario the mobile application invokes Hydra semantic service selection. Thus, it needs to be a part of Hydra network. For this we deployed on the cell phone the Network Manager Lite version for Android [43] developed in Hydra (see Figure 14). The standalone application has been generated with Limbo [44], an efficient compiler for deploying Web Services on resource-constrained devices. The Network Manager middleware component represents a sophisticated network middleware component that facilitates through applying SOAP tunnelling the communication beyond NATs and firewalls (see D5.10 [8] section 3.2) and solves the thrust in pervasive systems when endpoints frequently change or disappear due to changing locations (mobility) [12]; it is the entry and exit point of information flowing in Hydra network (see D3.9 section 7.3). By this, we have supported the user while interacting with his cell phone to better cope with the challenges in his pervasive environment.



Network Manager Full Version for Android

Network Manager for Android

RDV Server: tcp://212.214.80.136:9101

Relay Server: http://212.214.80.136:910(

Service port: 8082

HID: 12.12.12.12

Description: NetworkManager:FullAndr(

Start Exit Reload

STATUS

... Parameters loaded ...

Figure 14 Network Manager deployed on G1 Android Phone

In the scenario the user interacts with the intelligent application based on Hydra middleware for supporting him to select the best-suitable, available *PlayVideo* service. At first, he invokes the process of searching for the best-suitable service (see Figure 15a).

Before initiating the search the user can optionally apply the extreme rule for a default property, e.g. in the scenario Lucy selects *Resolution* property as extreme rule (see Figure 15b).

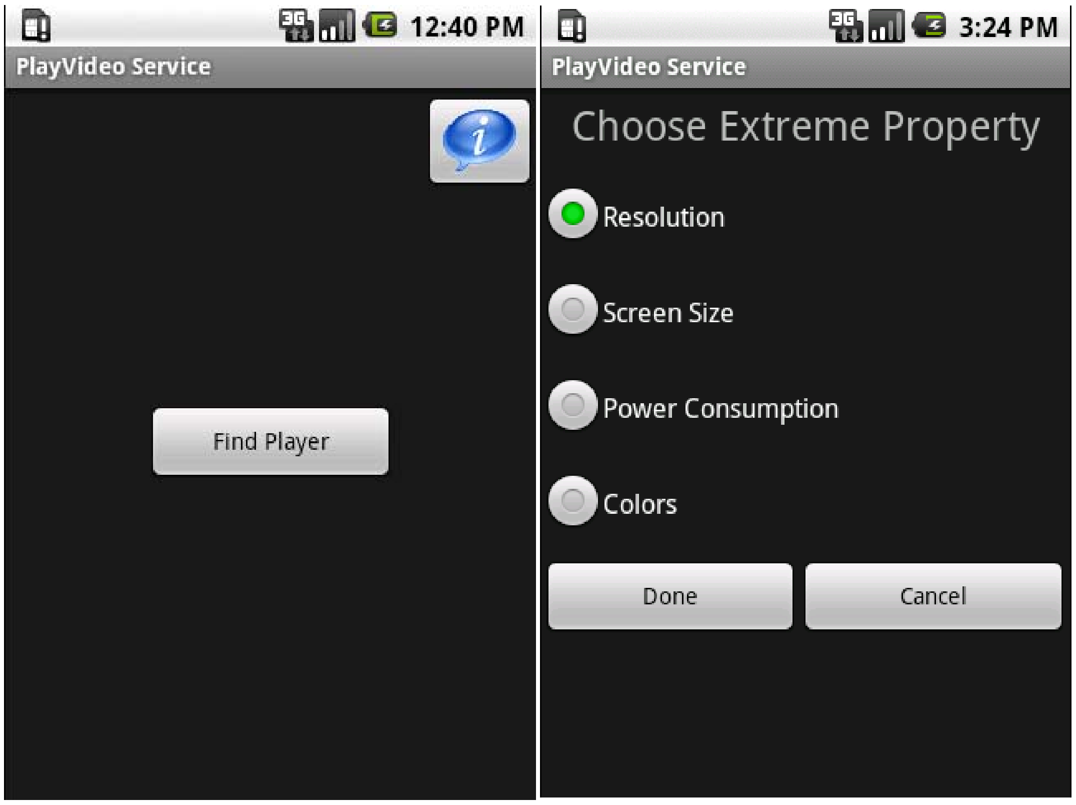


Figure 15 Find Player GUI (a), Choose Extreme Property (b)

When the search is initiated, the application starts to send users' requests to the underlying pervasive middleware architecture (see next section). The search may take some while as the processing of querying, computing, decision-making, etc. takes up some time. In the meanwhile the user will get some search feedback (see Figure 16a).

After the semantic service selection has been executed, the best three results for suitable-services are presented on the UI as a simple ranking list. Further, the fitting average rate is displayed (see Figure 16b). The user has the choice to select among the three best-suitable *PlayVideo* services.

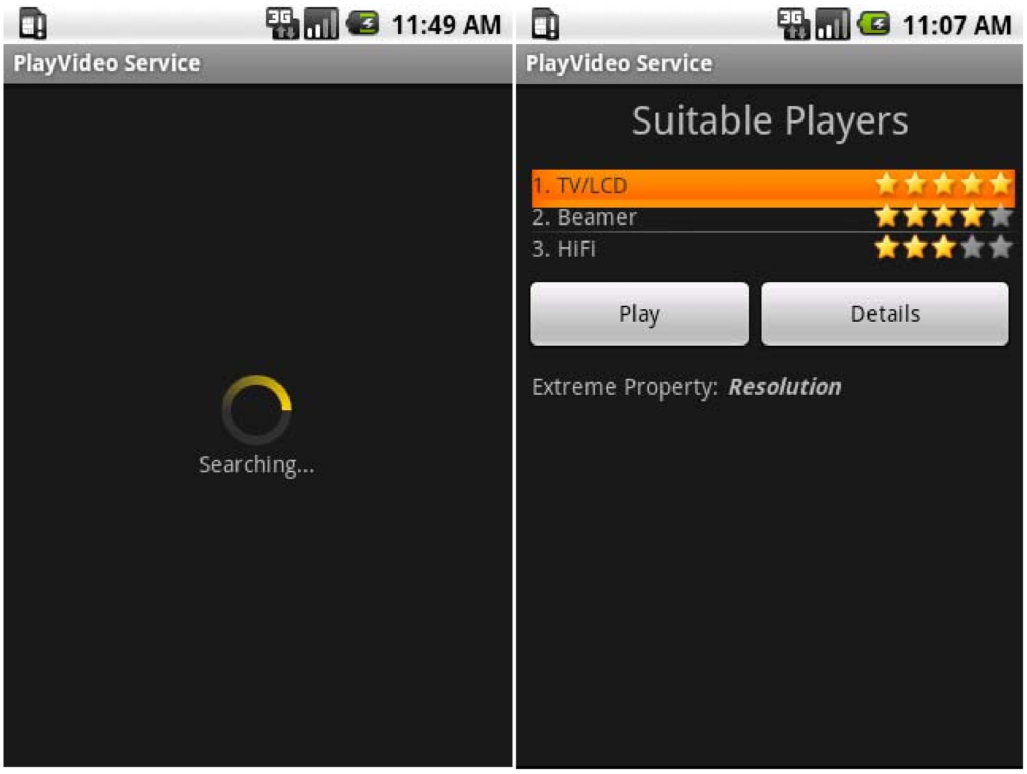
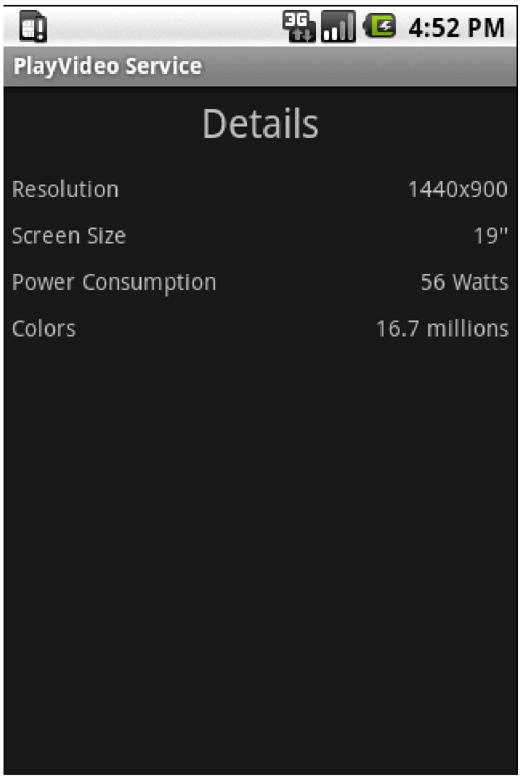


Figure 16 Searching Services (a), Service Ranking List (b)

For the end-user the exact average fitting rates are irrelevant. In fact, in the scenario it happens to be that the TV LCD screen places first, although it has a considerable lower fitting average rate (59,5%) than the beamer (67,52%)—see the next section for the computation of the fitting rates. To get a better understanding the user has the possibility to view the specific service device-based properties (see Figure 17).


4:52 PM

PlayVideo Service

Details

Resolution	1440x900
Screen Size	19"
Power Consumption	56 Watts
Colors	16.7 millions

Figure 17 Service Property Details

6.3.4 Pervasive Application Architecture

Figure 18 depicts the architecture of our scenario. The zigzag lines imply Web Service calls going over the Hydra P2P overlay network; for the sake of simplicity we have skipped the calls of SOAP-Tunnelling and Network Manager in the flowing routes.

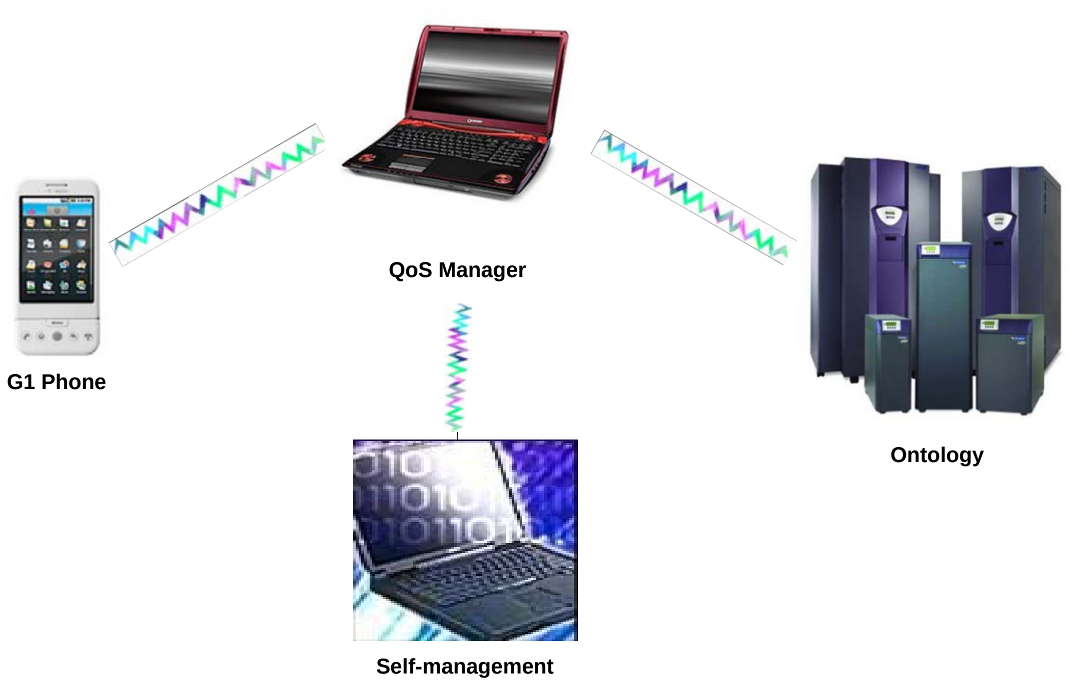


Figure 18 Pervasive Middleware Application Architecture

When the mobile application is invoked for finding suitable players, the application starts to send over Hydra network user request to the application-specific Hydra QoS Manager. Accordingly, the QoS Manager forms SPARQL queries for retrieving the domain specific service device-based properties—by the use of its internal rule engine component. These formed queries are sent to Hydra Ontology Manager for querying Hydra ontology for suitable services. After processing the Hydra Ontology Manager returns back the appropriate service property data values to QoS Manager. Now, QoS Manager can process its semantic service selection algorithm. The queried service properties and the decision results made through the engines inside QoS Manager may all be forward to self-Management Manager to preserve as the basis for self-configuration and self-adaptation.

6.3.5 Semantic Service Selection Algorithm

In the scenario the *Resolution* property is considered as extreme rule, i.e. Lucy prefers the largest one, so the standard for *Resolution* property is `most`.

Other three properties will remain set default as `less` power consumption, `more` colours and `more` screen size. Those requests with property name and standard will be forwarded to the computation engine in sets of **`Strings`**.

Thereon, the computation engine will calculate the fitting for each property and make the appropriate ranking lists depicted below:

Resolution	Property Data Values (in %)
1. TV LCD	100%
2. Beamer	55.55%

3. Hi-fi	0%

Screen Size	Property Data Values (in %)
1. Beamer	100%
2. TV LCD	38%
3. Hi-fi	0%

Power Consumption	Property Data Values (in %)
1. Hi-fi	100%
2. Beamer	14.54%
3. TV LCD	0%

Colours	Property Data Values (in %)
1. Beamer	100%
2. TV LCD	100%
3. Hi-fi	0%

Table 9 Scenario's Properties Fitting Rate

So, the final ranking list after calculating the average fitting rate for each device-based *PlayVideo* service will be:

Average Fitting Rate for *PlayVideo* Services	Property Data Values (in %)
1. Beamer	67,52
2. TV LCD	59,5
3. Hi-fi	25

Table 10 Scenario's Average Fitting Rates

An extreme rule, a user has defined, always meets the highest priority. In the scenario, for the resolution the extreme rule has been applied, and thus the TV LCD screen get the first place. Although the Beamer gets the highest average fitting rate, it places second rank, as the TV LCD screen has a higher resolution.

Final Ranks for *PlayVideo* Services	Property Data Values (in %)
1. TV LCD	59,5
2. Beamer	67,52
3. Hi-fi	25

Table 11 Scenario's Ranking List

6.4 Outcome

In this chapter we have applied the semantic service selection algorithm of QoS enabled Hydra middleware for a real-world scenario in the home automation domain. The user was equipped with a G1 Android cell phone that has running on it a Lite version of Hydra Network Manager for resource-constrained devices, and is thus able to interchange data with the Hydra network. By this scenario we have proven the concept of intelligently supporting the user to select the best-suitable service—out of a range of similar services—fulfilling his specific needs. However, this example also shows how complex such QoS enabled semantic service selection process can be, as you need to take into consideration a lot of different variables, and the algorithm requires some preparation and passes through different phases of calculation. And besides that, this scenario just reflects a rather simple use case, as it is only constituted on an application domain that provides similar *PlayVideo* services with only four service properties.

By hindsight while reflecting on the above described scenario and on our approach, i.e. how we have coped with the service selection problem, we can think of two main questions for future work in regard of the Hydra QoS enabled middleware:

1. To examine the scalability when applying Hydra semantic service selection for applications, which consider much more service properties than this simple, scenario does. We need to validate how our semantic service selection grasps when having to deal with a greater number of variables, similarities among data values etc.
2. The way of computation has rather been performed statically due to the specifics for each application domain. However, to conduct studies over a longer period, and to conclude if a rather dynamical computation would be possible or to derive guidelines for less static computation poses a delicate challenge.

7 Conclusions

The Quality of Service prototype is the result of the work of WP4 within Hydra during the third iteration of the project. However, we have found out that this work strongly relates to application specific work that belongs to WP6.

The starting points of this work are D4.5 [3], D4.7 [4], and D8.3 [5], and the lessons learned from the second year prototype. The QoS related middleware components will be provided this month (M36) by WP4 in order to have a complete Hydra QoS enabled middleware distribution.

The QoS related tools and middleware components have been updated to introducing new functionality. Furthermore, reviewing existing QoS related functionality has been revised and adapted as results of Lessons Learned from the second year prototype. By this document we have provided the relevant implementation description of these components.

The Ontology Manager allows developers to define concepts in Hydra Ontology in order to QoS enable Hydra services. Through an accessible web interface specific QoS property data values can be retrieved by the formulation of queries, e.g. SPARQL [2] queries.

The QoS Manager has been updated according to the Lessons Learned from the second year prototype providing new functionalities, such as:

- *Semantic Service Selection* resulting the best-suitable services while considering either the best average fitting rate or an extreme criterion that has been defined for a specific QoS property.
- Monitoring constantly changes of static and dynamic QoS properties that will be forwarded to self-management component for QoS-based self-adaptation, configuration, and self-protection.

In order to provide a comprehensive pervasive middleware Hydra needs to facilitate QoS for Web Services. At first, Hydra developers must be able to QoS enable Hydra services by defining concepts in Hydra ontology. And second, developers must have the possibility to consume QoS aware Hydra services.

One of main lessons learned from the second year prototype was that selecting the best-suitable service personates a difficult task for service consumers. Therefore, we have extended Hydra middleware with QoS semantic service selection. The service selection supports to select the best-suitable services out of a range of Hydra services providing same functionality, though they differ in terms of quality-of-service (see chapter 3).

Referring to relevant QoS properties we have found out that ‘cost’ and ‘power consumption’ are viewed as important QoS properties, besides performance related properties. Service consumers care what a service may cost over a certain time period and in respect to energy efficiency the power consumption of devices plays an important role (see sections 3.2 and 3.3.1).

Furthermore, self-management component requires a regular update of changed QoS properties in order to perform QoS-based self-adaption, configuration, and self-protection (see chapter 4). This issue is described in detail in D8.4 [28]. For this purpose, the QoS Manager offers an accessible web interface returning back a set of changed QoS property data values.

In regard to QoS enabled Hydra semantic service selection we think to consider a further mechanism that facilitates to assign to QoS properties different weights respectively, i.e. property A counts twice as much as property B, may represent an interesting alternative to calculating the average fitting rates or applying an extreme criterion (see section 5).

Moreover, we like to stress that context as additional variable must be considered during semantic service selection (see section 5.3). Although, context and QoS may overlap in the definition of service properties, the scenario described in D4.5 [3] section 4.2.1 shows that combining QoS and ‘context awareness’ during semantic service selection may be worthwhile.

Not least, we have shown in chapter 6 how QoS integration of Hydra middleware can be applied for a real-world scenario in the home automation domain. By this, we have exemplified the required steps of applying Hydra QoS enabled semantic service selection for a specific application domain considering a predefined set of QoS properties. The process is certainly complex, even though this scenario reflects a rather simple use case, as it comprises: the formulation of (SPARQL) queries for retrieving QoS property data values, a set of computation methods in order to calculate fitting rates of QoS properties, and two different ways of decision making (average fitting rate or extreme criterion).

8 References

1. Eisenhauer, M., *Hydra Description of Work (DoW) Version 7.28* 30th June 2008.
2. Prud'hommeaux, E. and A. Seaborne. *SPARQL Query Language for RDF*. 15 January 2008; Available from: http://www.w3.org/TR/rdf-sparql-query/.
3. Scholten, M. and L. Shi, *Hydra Project Deliverable D4.5: "Quality-of-Service Enabled HYDRA Middleware", Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891*. 3rd July 2008.
4. Fernandes, J., et al., *Hydra Project Deliverable D4.7: "Embedded service DDK prototype and report", Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891*. 24th February, 2009.
5. Ahlsen, M., et al., *Hydra Project Deliverable D8.3: "Integrated Platform and Report for SDK Prototype", Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891*. 30th May 2008.
6. *Hydra Project Deliverable D3.9: "Updated System Architecture Report v1_1", Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891*. May 2008.
7. Zhang, W. and M. Ingstrup, *Hydra Project Deliverable D4.8: "Self-* properties DDK prototype and report", Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891*. 29th December 2008.
8. Milagro, F., et al., *Hydra Project Deliverable D5.10: "Wireless Network DDK Prototype", Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891*. 28th February 2009.
9. Braden, R., D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture*. 1994; Available from: http://www.ietf.org/rfc/rfc1633.txt.
10. Blake, S., et al. *An Architecture for Differentiated Services*. 1998; Available from: http://www.ietf.org/rfc/rfc2475.txt.
11. Alonso, G., et al., *Web Services*. 1 ed. 2003: Springer. 354.
12. Satyanarayanan, M., *Pervasive Computing: Vision and Challenges*. 2001, unknown.
13. Mani, A. and A. Nagarajan. *Understanding quality of service for Web services*. 2002 Available from: http://www.ibm.com/developerworks/library/ws-quality.html.
14. KangChan Lee, P., Electronics and Telecommunications Research Institute, et al. *QoS for Web Services: Requirements and Possible Approaches*. 25th November 2003; Available from: QoS for Web Services: Requirements and Possible Approaches.
15. International Organization for Standardization, *Software engineering - product quality*, in *ISO/IEC 9126-1. Technical Report, I*. 2001.
16. *OWL Web Ontology Language Guide*. 2004; Available from: http://www.w3.org/TR/owl-guide/.
17. Makripoulias, Y., et al., *Web Service discovery based on Quality of Service*, in *Proceedings of the IEEE International Conference on Computer Systems and Applications*. 2006, IEEE Computer Society.
18. Naveen, S., P. Massimo, and S. Katia, *Adding OWL-S to UDDI, implementation and throughput*. 2008, unknown.
19. Serhani, M.A., et al., *A QoS Broker Based Architecture for Efficient Web Services Selection*, in *Proceedings of the IEEE International Conference on Web Services*. 2005, IEEE Computer Society.
20. David Martin, S.I.e., et al. *OWL-S: Semantic Markup for Web Services* 22th November 2004; Available from: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.
21. Bell, M., *Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture*. 25th February 2008: Wiley 384
22. Kritikos, K. and D. Plexousakis, *OWL-Q for Semantic QoS-based Web Service Description and Discovery*, in *First International Joint Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web*. 11th November 2007. p. 14.
23. Wang, X., et al., *A QoS-aware selection model for semantic Web services*, in *4th Int. Conf. on Service Oriented Computing*. 2006.
24. Papaioannou, I.V., et al., *A QoS Ontology Language for Web-Services*, in *Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 01*. 2006, IEEE Computer Society.
25. Fujii, K. and T. Suda, *Semantics-based context-aware dynamic service composition.* ACM Trans. Auton. Adapt. Syst., 2009. **4**(2): p. 1-31.

26. Fujii, K. and T. Suda, *Semantics-based dynamic service composition.* IEEE Journal on Selected Areas in Communications, December 2005. **23**(12): p. 12.
27. Scholten, M., M. Eisenhauer, and A. Zimmermann, *Hydra Project Deliverable D2.5: "Initial Requirements Report", Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891*. 14th February 2007.
28. Fernandez, J., et al., *Hydra Project Deliverable D8.4: "Integrated Platform and Report for DDK Prototype", Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891*. 30 April, 2009.
29. Mokhtar, S.B., et al., *Amigo Project Deliverable D3.2: "Amigo middleware core: Prototype implementation and documentation", Contract No. IST-2004-004182*. March 31th, 2006.
30. Kritikos, K. and D. Plexousakis, *Semantic QoS-based Web Service Discovery Algorithms*, in *Proceedings of the Fifth European Conference on Web Services*. 2007, IEEE Computer Society.
31. *Java Sun*. The Source for Java Developers Copyright 1994-2009 Available from: http://java.sun.com/.
32. *OSGi Alliance*. OSGi™ - The Dynamic Module System for Java™; Available from: http://www.osgi.org.
33. *OSGi Service Platform*. Core Specification, Release 4, Version 4.1. 2007: OSGi Alliance.
34. *Apache Tomcat*. Copyright © 1999-2009; Available from: http://tomcat.apache.org/.
35. (2007) *IBM WebSphere Service Registry and Repository, Version 6.1* WebSphere Software.
36. *OSGi Declarative Services*. OSGi Service Platform, Service Compendium, Release 4
Version 4.1. 2007: OSGi Alliance.
37. *WinDump: tcpdump for Windows*. 1st December, 2006; Available from: http://www.winpcap.org/windump/.
38. Hansen, K.M., W. Zhang, and M. Ingstrup, *Hydra Project Deliverable D4.3: "Self-* properties SDK prototype and report", , Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891*. February 3, 2008.
39. Bellotti, V. and K. Edwards, *Intelligibility and accountability: human considerations in context-aware systems.* Hum.-Comput. Interact., 2001. **16**(2): p. 193-212.
40. Zhang, W., et al., *Hydra Project Deliverable D3.8: "Context-awareness Report", Integrated Project, SO 2.5.3 Embedded systems, Contract No. IST 2005-034891.* May 2008.
41. Abowd, G.D., et al., *Towards a Better Understanding of Context and Context-Awareness*, in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. 1999, Springer-Verlag: Karlsruhe, Germany.
42. Dey, A.K., *Understanding and Using Context.* Personal Ubiquitous Comput., 2001. **5**(1): p. 4-7.
43. *Android Homepage*. 2008; Available from: http://www.android.com/.
44. Hansen, K.M., et al., *Flexible Generation of Pervasive Web Services Using OSGi Declarative Services and OWL Ontologies*, in *Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference*. 2008, IEEE Computer Society.