Contract No. IST 2005-034891

Hydra

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

D3.4 Initial architectural design specification

Integrated Project
SO 2.5.3 Embedded systems

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

Published by the Hydra Consortium **21.Dec 2007 - version 1.0**
Coordinating Partner: C International Ltd.

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

Dissemination Level: Confidential

Document file: D3.4 Initial architectural design specification v1.0.doc

Work package: WP3 – Architecture design specification

Tasks: T3.2 – System & software architecture analysis
T3.3 – Architectural design

Document owner: Fraunhofer FIT

Document history:

Version	Author(s)	Date	Changes made
0.1	Markus Eisenhauer, Christian Prause, Alexander Schneider, Marius Scholten, Andreas Zimmermann	05-11-2007	First version for peer review
0.2	Christian Prause	10-11-2007	Adding the development chapter
0.3	Andreas Zimmermann	10-12-2007	Adding the lessons learnt chapter
0.4	Marius Scholten	12-12-2007	Adding the QoS Manager
0.5	Andreas Zimmermann	14-12-2007	Ready for internal review
0.6	Andreas Zimmermann	19-12-2007	Review comments integrated
1.0	Andreas Zimmermann	21-12-2007	Final Version

Internal review history:

Reviewed by	Date	Comments
Peter Rosengren, CNET	17-12-2007	
Klaus Marius Hansen, UAAR	17-12-2007	

Index:

1. Executive Summary ... 5

2. Introduction ... 6

3. Stakeholder and Scenario Analysis ... 7

3.1 Stakeholder Analysis ... 7
3.1.1 Facility owner (may be operator) ... 7
3.1.2 Facility manager (may be operational manager) ... 7
3.1.3 System integrator and solutions provider ... 7
3.1.4 End-users (Private tenants/Commercial tenants) ... 8
3.1.5 Maintenance crew ... 8
3.1.6 External providers ... 8
3.1.7 Social and government service provider/requester ... 8

4. Methodology ... 9

4.1 Software Architecture Design Fundamentals ... 9
4.1.1 Requirements & Architecture ... 9
4.1.2 Viewpoints & Views ... 9
4.2 Software Architecture Design Process ... 9
4.2.1 Architecture Definition Activities ... 9
4.2.2 Viewpoint Catalogue ... 12
4.2.3 Architectural Perspectives ... 13

5. General Design Considerations ... 14

5.1 Distributed vs. Centralized Approach ... 14
5.2 Providing Redundancy ... 14
5.3 Simplicity vs. Complexity ... 15
5.4 Middleware ... 15

6. Software Architecture ... 17

6.1 Overview ... 17
6.2 Functional View ... 17
6.2.1 Device Elements ... 18
6.2.2 Application Elements ... 32

7. Development View ... 54

7.1 Overview ... 54
7.2 Module Organization ... 54
7.3 Configuration Management ... 55
7.4 Development Decisions ... 56
7.5 Test Plan ... 57
7.5.1 Continuous Integration ... 58
7.5.2 Code Inspection ... 59
7.5.3 Relation to the Test Plan defined in D8.1 ... 60

8. Lessons Learnt from the First Iteration ... 61

8.1 Overview ... 61
8.2 Concept Prototype ... 61
8.3 Categorization of Lessons learned ... 62
8.3.1 Communication and Networking ... 62
8.3.2 Event Management ... 62
8.3.3 Interoperability ... 63
8.3.4 The Use of Web Services ... 63
8.3.5 Use of Ontologies ... 63
8.3.6 Separation of Application and Middleware ... 64
8.4 Implications on the Software Architecture ... 64
8.4.1 Introduction of an Information View ... 64
8.4.2 Introduction of a Common Deployment View ... 65

8.4.3 Improved Service-Oriented Architecture Approach 66
8.4.4 Integration of Context-Awareness 66
9. Guidelines and Constraints 68
9.1 Guidelines 68
9.2 Constraints 68
9.2.1 General Constraints...68
9.2.2 Domain Specific Constraints 69
10. Conclusion 70
11. List of Figures.. 71
12. References 72

1. Executive Summary

The Hydra project is researching and developing a middleware for heterogeneous physical devices in a distributed architecture. The goal is to develop a middleware that is 'inclusive' which means that it will be possible to enable any device to be detectable and usable from a Hydra application. It will also deliver development tools for solution providers of ambient intelligent applications using such devices. The complementary goal is also to develop tools for device producers to enable their devices to be part of an ambient intelligence environment.

For the overall architectural design process we use an iterative approach which is based on the work of Rozanski and Woods [Rozanski & Woods, 2005] and the architectural descriptions are in line with the IEEE 1471 standard.

The deliverable starts with an introduction to the stakeholders and a scenario from the building automation domain which is the focus in the first iteration of the project. Then we describe the used methodology and what general design considerations have been identified.

Chapter 6 follows with a description of the architecture is based on a functional viewpoint of the middleware in which we describe the identified layers and components and how they relate to each other. We describe the different components in detail and some more details of the sub-components. In addition to this we have identified the requirements which have an implicit or explicit relationship to the component.

In the following chapter we present a first scenario which we have researched in detail to explain what could be done with the architecture. After describing the scenario we present a set of sequence diagrams explaining the message exchange between the components.

Finally we show what kind of guidelines and constraints have been identified and in what way we want to extend this in future iterations.

This deliverable is the basis for the work in workpackages 4 to 7.

2. Introduction

The Hydra project is researching and developing a middleware for heterogeneous physical devices in a distributed architecture. In a distributed computing system, middleware is defined as the software layer that lies between the operating system and the applications on each site of the system. The goal is to develop a middleware that is 'inclusive' which means that it will be possible to enable any device to be detectable and usable from a Hydra application. It will also deliver development tools for solution providers of ambient intelligent applications using such devices. The complementary goal is also to develop tools for device producers to enable their devices to be part of an ambient intelligence environment.

The purpose of this deliverable is to explain the architecture in detail and what methodology was used to develop the architecture. The architecture is based on the deliverable D2.5 Initial Requirements Report because there the goals and needs of involved stakeholders and of our target users (the application developers) were collected. The challenges we face in this project are manifold. We have a highly complex mix of people (diverse set of stakeholders and domains) and technologies and a multi-dimensional problem to solve in that we want to be as flexible as possible to support as many future scenarios as possible. But there are trade-offs to make between the imaginable possibilities and the usability, maintainability and extendibility of the middleware. Those trade-offs are hard to make but are absolutely necessary and we will highlight important design choices and explain the reasons behind them.

We will use an iterative process because the complexities of the project can be handled only in this way. We will need to revise the architecture during the runtime of the project and an iterative approach ensures that we still make good progress. In addition to this we will continuously update the requirements and those changes need to be reflected in the middleware again and so on.

This deliverable will be the basis for the work in the workpackages 4 to 7 because here we will define the overall architecture and the components that will be used in these workpackages. We want to make the middleware extensible so we need to define clear and consistent interfaces between the components so that the vision of replacing a component by plugging in another component with a compatible interface can be realized.

3. Stakeholder and Scenario Analysis

3.1 Stakeholder Analysis

In the first iteration of the Hydra project we analyzed the Building Automation domain so in the following chapters only stakeholders from this domain will be presented. The two other domains will follow in the next iteration.

For the definition of requirements traditionally the user was taken as a reference. One could argue what exactly the term user includes but it is obvious that not only the requirements of the user have to be analyzed. The software has to be developed, run, administered, maintained, monitored and abides to certain standards or regulations. Each of these aspects is of interest to different people which not necessarily are using the system at all. We refer to these groups of people as stakeholders. We use the following definition (taken from IEEE 1471):

> **Stakeholder**: An individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.

We have identified seven types of stakeholders in our analysis of the Building Automation scenario. Each stakeholder has interests and concerns, which influence the requirements and also the architecture design. We will continuously check if the architectural design choices reflect also these interests and concerns.

3.1.1 Facility owner (may be operator)

The facility owner's responsibilities are to provide safe, secure and comfortable premises complying with the service level agreement that is explicitly stated in his contract with the tenants. He has to provide smart house services which he could outsource to other companies. He also has to obey regulatory standards e.g. laws by the government or apply industry standards e.g. not to invalidate insurance contracts.

Since smart homes are mainly found in high-end apartments the facility owner's goal will be to be the most attractive provider of private or commercial space and ensure that he has a competitive advantage against competitors. He wants to continuously get provisions of smart home service bundles at competitive or market rates.

3.1.2 Facility manager (may be operational manager)

The facility manager's responsibilities are ensuring a 24/7 continuity of the services in public and private areas. He has to provide security in the facility. His goal is to ensure an efficient and effective delivery of 24/7 maintenance and security provision according to the SLA at the lowest possible cost. For this he needs to make sure that he has a robust and reliable supplier base of subsystem services.

3.1.3 System integrator and solutions provider

The system integrators responsibilities are to develop a technically sound application for integration and interoperability of the smart house backbone and available devices and services as well as the availability of telemetric services. The developers of the system integrator will use the Hydra middleware intensively. He needs to perform continuous upgrade and extension services and ensure compliance with national and global standards and regulations. His goal is to deliver the implementation of the application to contract cost and time effectively. The provider wants to ensure business continuity by way of risk litigation.

3.1.4 End-users (Private tenants/Commercial tenants)

The responsibilities of the tenants are to behave conformant to their contractual commitments to the facility owner or facility manager and regulatory bodies. The tenant's goal is to avail themselves of the smart house beneficially and securely and without compromising privacy.

3.1.5 Maintenance crew

The maintenance crew has to monitor the subsystem states and react responsibly to provide maintenance according to the SLA. They are also responsible to deliver diagnostic and logging information to the smart house data repository. Their goal is to fulfil the SLA´s contractual obligations and ensure cost effectiveness in monitoring and maintenance.

3.1.6 External providers

An external providers can be a subsystem provider, a content provider, a remote service provider, a network service provider, etc. This also includes the manufacturers who will eventually design, develop and build Hydra-enabled devices which are used by the system integrator´s developers.

The responsibility of external providers of systems and services is to provide the contracted service in full compliance with the SLA and regulatory standards. Their goal is to deliver the services according o the SLA most cost effectively and ensure business continuity and profitability through risk litigation.

3.1.7 Social and government service provider/requester

The responsibilities are to set and enforce the deployment of regulatory standards and to seek to deploy the smart house technology at the service of the constituency most cost effectively without compromising end-user security and privacy. Their goals are to ensure that appropriate standards and regulatory procedures are evolved conforming to the governance and policing of quality of service in the smart house services sector. They also want to select and procure the most cost effective smart house services.

4. Methodology

4.1 Software Architecture Design Fundamentals

We have based our process on the standard IEEE 1471 "Recommended Practice for Architectural Description of Software-Intensive Systems" which defines core elements like viewpoint and view. It also describes that stakeholders need to be involved and how to apply stakeholders needs to the architecture. This will be supported by the introduction of "architectural perspectives" which was introduced by Rozanski and Woods [Rozanski, 2005].

4.1.1 Requirements & Architecture

We have established a process to gather requirements in a structured way as is laid out in deliverable D2.5 Initial Requirements report. For this we have conducted discussion rounds with focus groups of expert developers which possibly will use the Hydra middleware. The requirements were then prioritized and a fit criterion selected which allows to measure if a requirement is met or not. We have not only deduced requirements from the focus group discussions but also from other sources e.g. standards, best practices, each partner´s experience and so on. In this way we made sure that we collect a broad range of requirements to reflect the wide range of stakeholders.

Requirements and architecture influence one another. Requirements are an input for the architectural design process in that they frame the architectural problem and explicitly represent the stakeholders needs and desires. On the other hand during the architecture design one has to take into considerations what is possible and look at the requirements from a risk/cost perspective.

4.1.2 Viewpoints & Views

The IEEE 1471 standard defines viewpoint and view as follows:

> A **viewpoint** is a collection of patterns, templates and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint, and guidelines and principles and template models for constructing its views.
>
> A **view** is a representation of all or part of an architecture, from the perspective of one or more concerns which are held by one or more of its stakeholders.

A viewpoint defines the aims, intended audience, and content of a class of views and defines the concerns that views of this class will address e.g. Functional viewpoint or Deployment Viewpoint.

A view conforms to a viewpoint and so communicates the resolution of a number of concerns (and a resolution of a concern may be communicated in a number of views).

4.2 Software Architecture Design Process

4.2.1 Architecture Definition Activities

Rozanski and Woods have based the architectural design process on the following definition:

"Architecture Definition is a process by which stakeholder needs and concerns are captured, an architecture to meet these needs is designed, and the architecture is clearly and unambiguously described via an architectural description." [Rozanski, 2005]

We have to consider a broad set of principles if the architectural design should be of good quality. We need to engage stakeholders to collect their concerns so the requirements can be balanced if there are conflicting or incompatible ones. The architectural design must allow for effective communication between all stakeholders and it must be structured to ensure continuous progress. Given the complexity of the project the design and also the process has to be flexible so we can

react quickly to changing requirements and environments. An architecture should be technology neutral but in the case of Hydra we have to ensure that it is applicable to a wide range of technologies because Hydra is inclusive.

The foundation for our process is the IEEE 1471 standard and we have used the process proposed by Rozanski and Woods which is aligned to this standard:

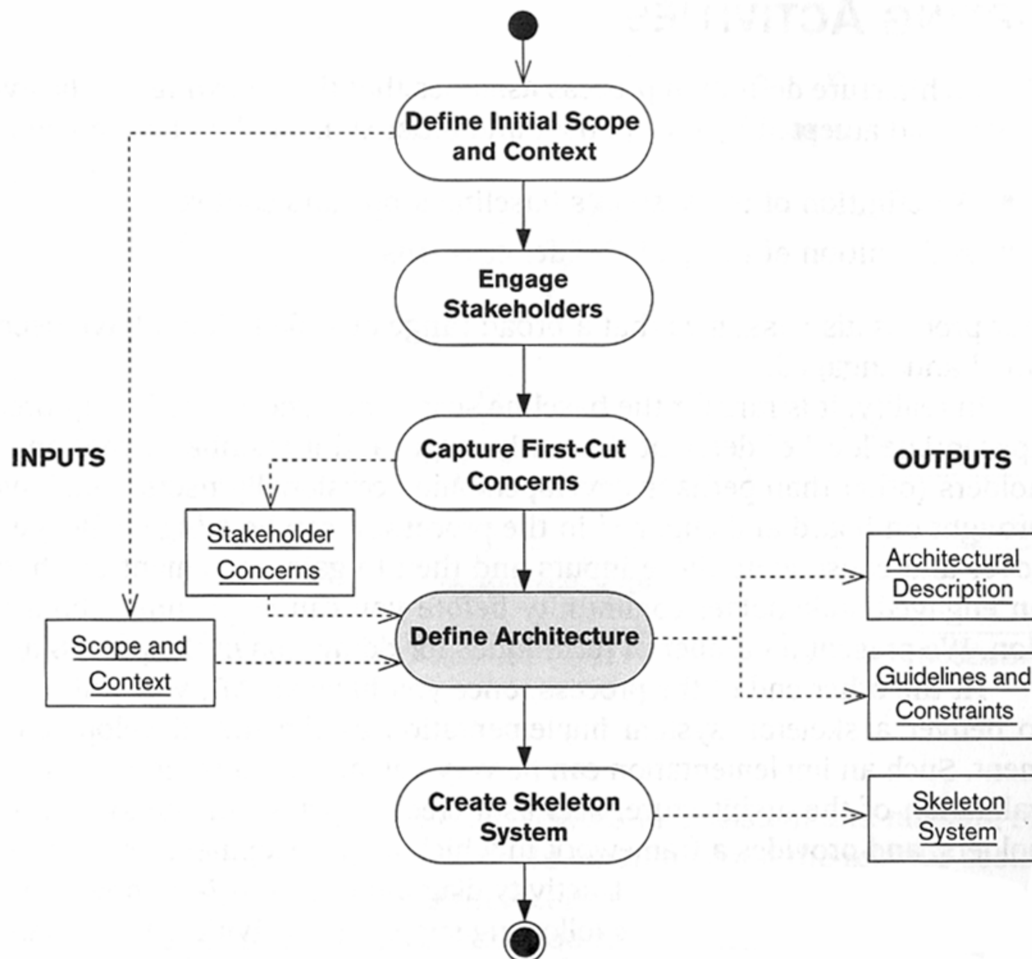

Figure 1 Architecture Definition Activities

The process implemented in the Hydra project clearly reflects this approach. We started with the initial scope and context and the involvement of stakeholders in the process of the scenario development in WP2 and the subsequent requirements process. The stakeholders were included to express their needs and desires and capture quality properties that increase the success of the middleware. Those requirements from the discussion rounds together with requirements from other sources are the input for the current architecture design phase where we create a first draft of the architectural description (AD). Based on this architectural description, the first prototype has been created, which can be seen as a skeleton system with minimal functionality on top. These development efforts revealed some experiences and lessons learnt, which in turn constitute a valuable source for the derivation of additional requirements and the revision of already existing ones.

The following diagram reflects the details of the process:

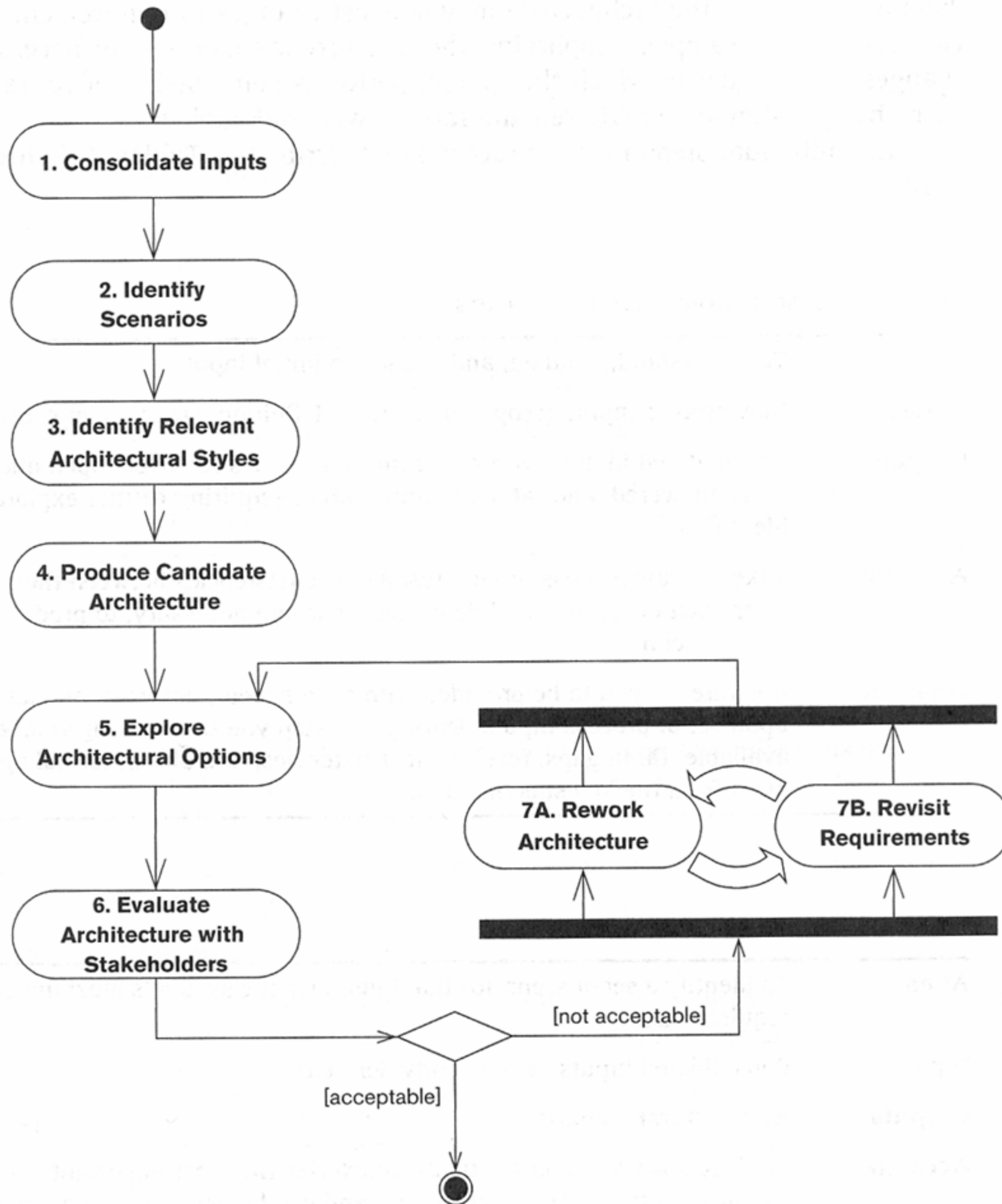

Figure 2 Architecture Definition Activities details

Steps 1 and 2 are reflected in the requirements process and steps 3 and 4 were basically defined by the DOW. In the DOW we have decided to implement a middleware based on a service-oriented architecture (SOA) and use web-services. With this as a framework the candidate architecture was set so we would only chose another architectural style if we would face insurmountable problems which is very unlikely.

The steps 5 to 7 (a and b) reflect our iterative approach on constantly refining the architecture and checking back with the stakeholders if the architecture meets their needs. After this iteration cycle the next steps of implementation and testing the revised architecture will follow but are not scope of this document.

4.2.2 Viewpoint Catalogue

The viewpoint catalogue proposed by Rozanski and Woods contains the following viewpoints:

Functional: The system´s functional elements, their responsibilities and primary interactions with other elements will be described. This is usually the most important viewpoint as it reflects the quality properties of the system and influences the maintainability, the extensibility and the performance of the system.

Information: Describes the way that information is stored, managed and distributed in the architecture.

Concurrency: Describes the concurrency structure of the system and identifies components that can be executed concurrently and how this is coordinated and controlled.

Development: Describes how the architecture supports the development process.

Deployment: Describes the environment that the system will be deployed into and also documents the hardware requirements for the components and the mapping of the components to the runtime environment that will execute them.

Operational: Describes how the system will be operated, administered and supported while it is running and strategies and conflict resolutions will be documented here.

The following diagram shows how the viewpoints relate to each other.

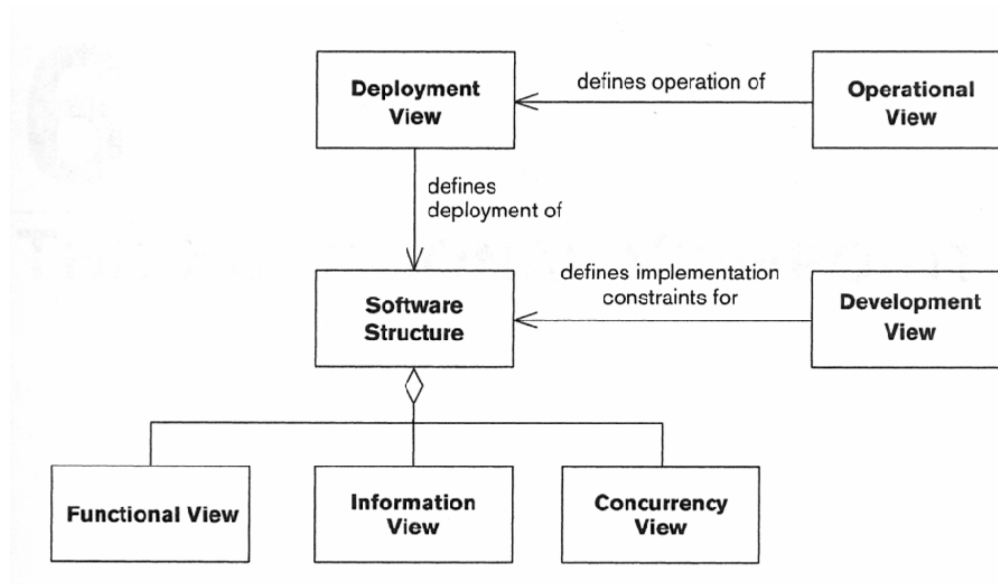

Figure 3 Viewpoint catalogue

During the course of the project this document will be continuously and successively extended by additional views.

4.2.3 Architectural Perspectives

The term "Architectural Perspectives" was coined by Rozanski and Woods.

> *An architectural* ***perspective*** *is a collection of* ***activities, checklists, tactics*** *and* ***guidelines*** *to guide the* ***process*** *of ensuring that a system* ***exhibits*** *a particular set of closely related* ***quality properties*** *that require consideration across a* ***number*** *of the system's architectural* ***views.***
>
> Rozanski and Woods, 2005

Figure 4 Definition of Architectural Perspective

The architectural perspectives ensure that quality properties are not forgotten in the process because the viewpoint and view approach per se does not explicitly consider those quality properties. But those properties are critical to the success of the project and to reflect them properly one usually needs cross-view considerations while the viewpoints are relatively independent.

The perspectives and the combination possibilities with views that Rozanski and Woods propose are shown in the following diagram:

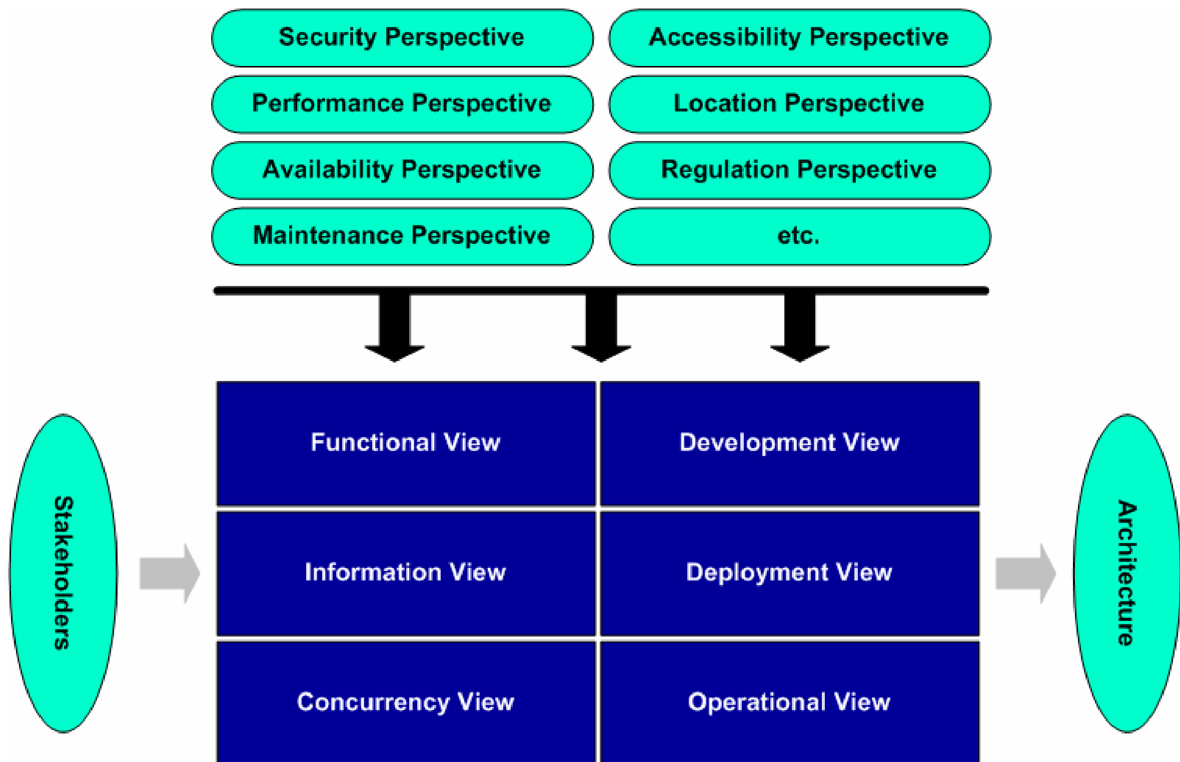

Figure 5 Architectural Perspectives and Views

They propose the perspectives security, performance, availability, maintenance, location, regulation etc. but not all combinations of perspectives and views are needed. Artefacts created according to those perspective/view combinations need to be carefully chosen to prevent "analysis paralysis" and getting lost in details. But these perspectives will be a good opportunity to check if all considerations have been done in the process. In later iterations we will choose the most appropriate combinations and apply the guidelines and checklists for the different perspectives.

5. General Design Considerations

Hydra systems spread over a large number of devices that again can spread over large spatial areas. For those highly distributed systems some basic design principals have to be considered. This chapter enumerates and describes an array of general design considerations that form the character and affect the Hydra architecture. Note that while some of these considerations may seem obvious in retrospect, the contribution of this section lies not only in identifying them, but also in illustrating how the design of the architecture addresses these issues. These issues influence implicitly but also explicitly the software development process of the Hydra middleware.

5.1 Distributed vs. Centralized Approach

Future Hydra devices combine several hard- and software components such as content, applications, displays, etc. required for the delivery of multi-purpose services. Users of such devices will share their content with other users either over a network or through other storage media. The Hydra project aims at an automatic sharing of content and information between several devices of a single user, but also among users. In addition, Hydra focuses on applications and services that will be deployed in environments, in which parts of the application need to be distributed. The distribution occurs on two different levels: on a conceptual level where information is distributed and on an implementation level where system components are distributed. A management of distributed components occurs in a centralized or decentralized manner.

A centralized approach is based upon a centralized component or server for several types of information and services, which provide requested information to the applications running on several devices. This approach decouples the acquisition of information (content, user-related information, context, device properties, etc.) from the processing of this information. These applications can actively request the desired information from the server or passively be notified about changes. The server collects all information from accordant acquisition components and provides it to interested applications. A centralized approach suffers from restricted scalability, in consequence of a maximum of applications that can be served by the server. In addition, the problem of privacy rises, since all user-related information is bundled and stored in one place. Instead of maintaining all information and services in one centralized place, a distributed approach holds the information at several places to avoid a potential bottleneck. Small devices maintain the information required by the application themselves and process it directly. This approach requires the device to have the capability to store and process all of the necessary data, which may not be efficiently achieved for a simple device with restrictions concerning space, weight, or energy consumption. The decentralized approach avoids the lacking scalability of the centralized approach and allows the user to control the way how their personal information is published and thus, their privacy is guaranteed. On the other hand, the sharing of information while preserving privacy is an issue.

5.2 Providing Redundancy

In distributed systems - independent of their degree of decentralization - a certain redundancy of contents and services must be assured to guarantee a certain level of robustness.

The Hydra middleware must guarantee robustness, independent of its degree of centralization. In centralized systems, servers are not only performance bottlenecks but also a weak point in terms of robustness. A faulty server can cause a failure of the whole system.

If information is distributed over multiple devices, the failure of a single information serving device is not as crucial to the functionality of the whole system as in a centralized system. Instead, the problem of finding and accessing information is relevant for the robustness in a decentralized system: Client devices must know the device, which holds specific information. Without a central server, this knowledge must be distributed over all devices as well as the information itself. Redundancy of content and services increases the probability, that clients find information in a distributed system.

An appropriate structuring of the Hydra system through a system architecture is essential. As a partitioning scheme for software, such an architecture separates concerns and directs the distribution of the architecture constituents.

5.3 Simplicity vs. Complexity

The paradigm of End-User Development (Fischer, 2002; Liebermann et al., 2006) aims at making systems that are easy to develop and empowers end-users to configure and compose the information technology according to their diverse and changing needs. At the core of End-User Development research is the question how to reduce the complexity the user is confronted with when adapting and configuring technology. Therefore, Mørch (1997) introduced three levels of complexity that avoid big jumps in complexity and address users at different stages of expertise and development skill. These levels allow users to

- select between predefined behaviours,
- compose a desired application out of existing modules, and
- fully access the code base of an application.

This property of avoiding big jumps in complexity to attain a reasonable trade-off is called the 'gentle slope of complexity' (MacLean et al., 1990; Dertouzos, 1997; Wulf and Golombek, 2001; Beringer, 2004). Users have to be able to make small changes in a simple way, while more complicated ones should only involve a proportional increase in complexity the user is confronted with. The software architecture of the Hydra system needs to achieve this gentle slope of complexity through the increase of the flexibility of the underlying technology. Object-oriented and component-based software paradigms allow for the introduction of different levels of complexity that address several expertise levels of a variety of users according to their specific roles.

For experienced developers of Hydra applications and services the code base would offer a large collection of reusable core software components. Programming abstractions will allow experts for a structured programming with well-known concepts through reducing the details of the underlying implementation. The configuration ability through a mark-up language enables designers of Hydra applications and services to tailor the arrangement of functional units of the software architecture to a concrete project. Changes in the configuration affect the function and performance of the desired application. For end-users the combination of the configurability with a corresponding set of tools provides the basis for the balance between full user control over the application behaviour and application autonomy. The appropriate position along the continuum between full user control and application autonomy will be dictated by the user's needs, situation and expertise.

5.4 Middleware

The concept of 'middleware' in distributed systems is often taken to mean "the software layer that lies between the operating system and the applications on each site of the system" (Krakowiak, 2003). Another characterization in terms of the ISO OSI stack (Day and Zimmerman, 1983) is that middleware provides protocols that run on top of the transport layer and that provides services to the application layer (Tanenbaum and Van Steen, 2007, p. 123) as shown in Figure 8.

Thus, application services such as graphical user interface support or domain-specific application functionality and transport-level services such as sockets are most often thought not to be part of middleware. The main functions of middleware are to

- provide *transparency* in various ways such as location transparency, access transparency, or failure transparency (ISO, 1995)
- provide high-level *programming interfaces* such as Remote Procedure Call, publish/subscribe, or reliable, ordered multicasting, and
- supply a set of *existing services* such as an Event Service, Security Service, or Transaction service for Corba or WS-Coordination, WS-Security, or WS-Notification implementations for Web Services.

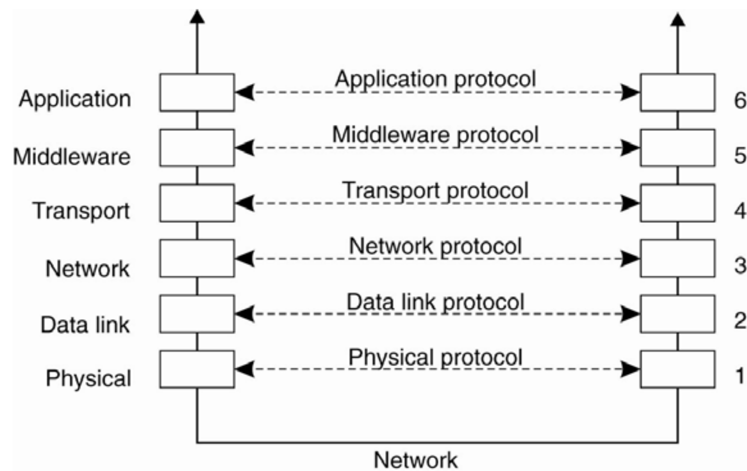

Figure 6 Middleware Layer

Douglas C. Schmidt (2002) elaborated a more detailed insight into the middleware layer and divided into the layers Host Infrastructure Middleware, Distribution Middleware, Common Middleware Services and Domain-Specific Middleware as shown in Figure 7. The Host infrastructure Middleware encapsulates and enhances native operation systems, and abstracts from sockets and provides higher-level abstractions (such as active objects), e.g., a virtual machine such as the Java Virtual Machine. The Distribution Middleware defines higher-level distributed programming models like for example CORBA or web services. The Common Middleware Services constitute higher-level domain-independent components for tasks such as event notification or logging. The Domain-Specific Middleware is tailored to specific system domain such as avionics or radar processing and manages issues like navigation management.

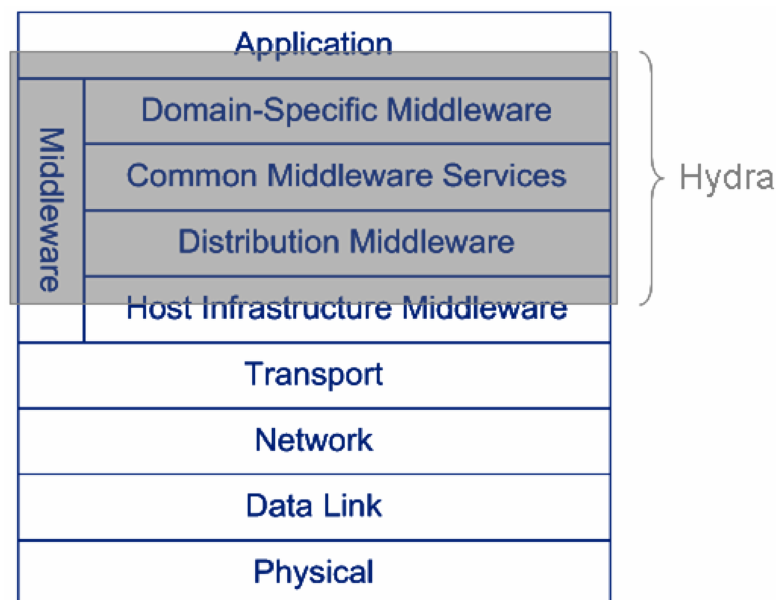

Figure 7 Generic Middleware Stack

Figure 7 also displays the areas of the generic middleware stack that the Hydra middleware will cover in future.

6. Software Architecture

6.1 Overview

This chapter describes the Functional viewpoint which is currently the main focus of our analysis. Other viewpoints like Information, Concurrency and Operational will be added in later iterations.

6.2 Functional View

The Functional Structure model is divided into two parts: Application Elements and Device Elements. Both elements differ in the following aspects:

- Resources available on the machine on which they are supposed to run (e.g., embedded platform vs. server platform)
- Intended purpose of the components (e.g., to support domain-specific development of applications or to support the development of application-independent devices)
- Target developer user (e.g., application developer or device developer)

Application elements describe components that are usually deployed on hardware which is performance-wise capable of running the application that the solution-provider creates. This means these components are meant to be run on powerful machines. They have been put together and configured to work together with other software in order to support a specific application such as building automation by a specific developer e.g. system integrator.

Device elements describe components that are usually deployed inside Hydra-enabled devices so we take into account that they could be deployed in small devices which have limited resources in terms of e.g. processing power or battery life. Those components have a limited set of functionality but could also be deployed on another machine acting as a proxy for e.g. a mote where it would be highly unlikely that those managers would ever be deployed on such a resource-limited device. They have been put on the device by a device manufacturer to provide certain functions irrespective of which application is using the device. The following diagram gives a structural overview of the middleware layers:

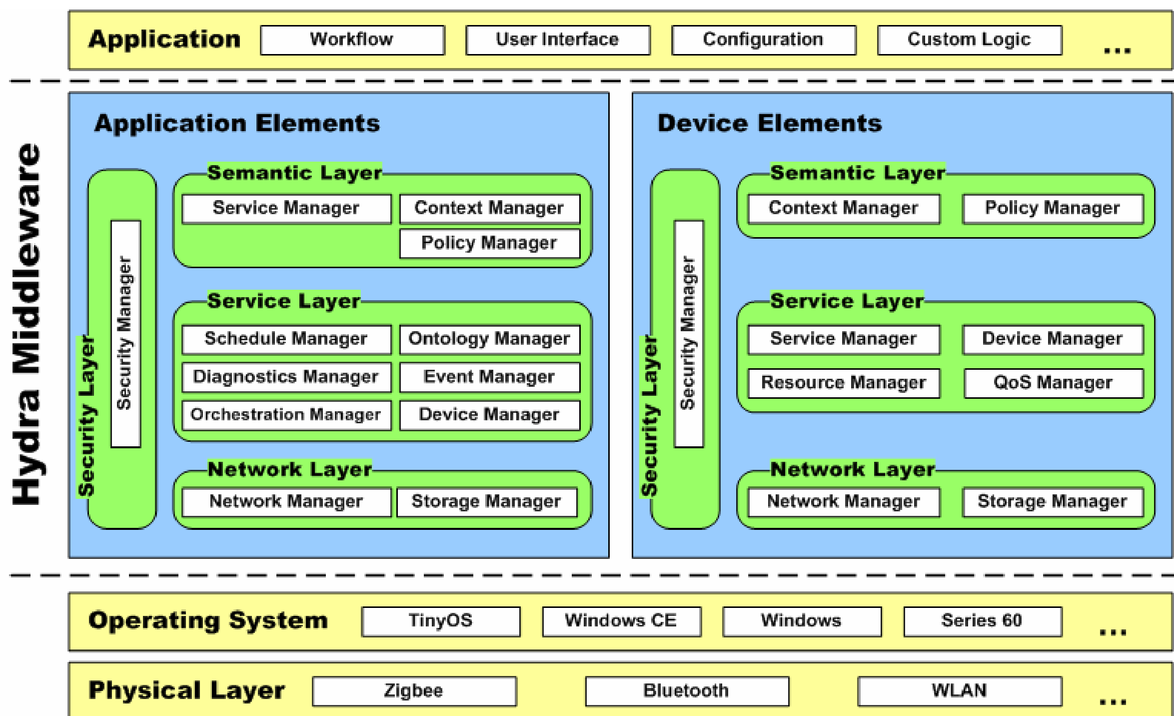

Figure 8 Software architecture layers

The Hydra middleware elements are enclosed by the physical, operating system and the application layer shown at the bottom and at the top of the diagram respectively. The physical layer realizes several network connections like ZigBee, Bluetooth or WLAN. The operating system layer provides functionality to access the physical layer and manages many other hardware and software resources and provides methods to access these resources. The application layer contains user applications which could contain modules like workflow management, user interface, custom logic and configuration details. These two layers are not part of the Hydra middleware. The middleware itself consists of three layers - the network, service and semantic layer. Each layer holds elements according to their functionality and purpose. Note, that some device elements have similar and thus similar named, counterparts among the application elements. Both, device and application elements, have a Security Manager. To express, that this manager is an orthogonal service, it is depicted vertical and covers all three middleware layers. The device and application elements are described in detail in the following chapters.

6.2.1 Device Elements

Here is a diagram explaining how the device elements are logically grouped and in the following chapters each element will be explained in detail:

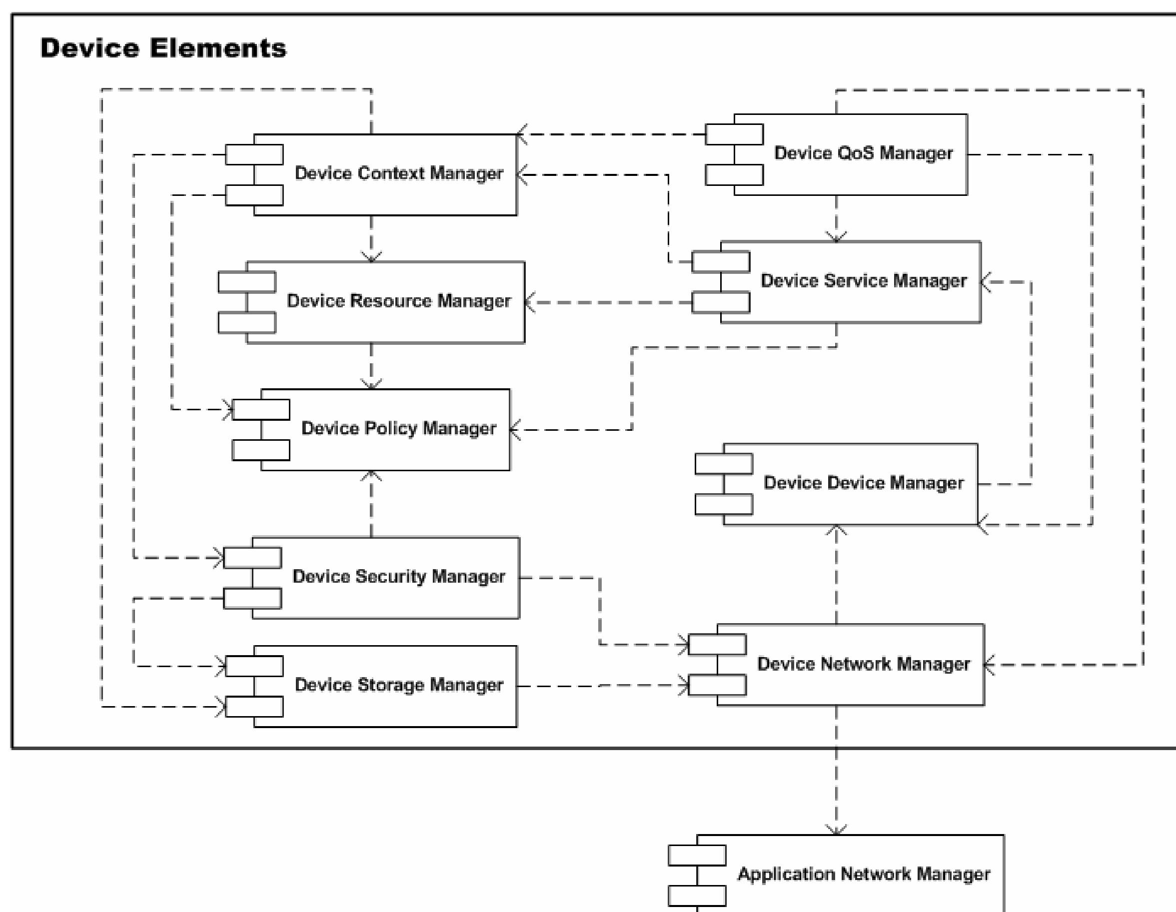

Figure 9 Overview of the Device Elements

6.2.1.1 Device Context Manager

The Device Context Manager is a component of an individual user's device which is normally carried around in his/her vicinity. The Device Context Manager is continuously aware of the current location and context the user is in. According to Dey (2001) "Context is any information that can be used to characterise 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.".

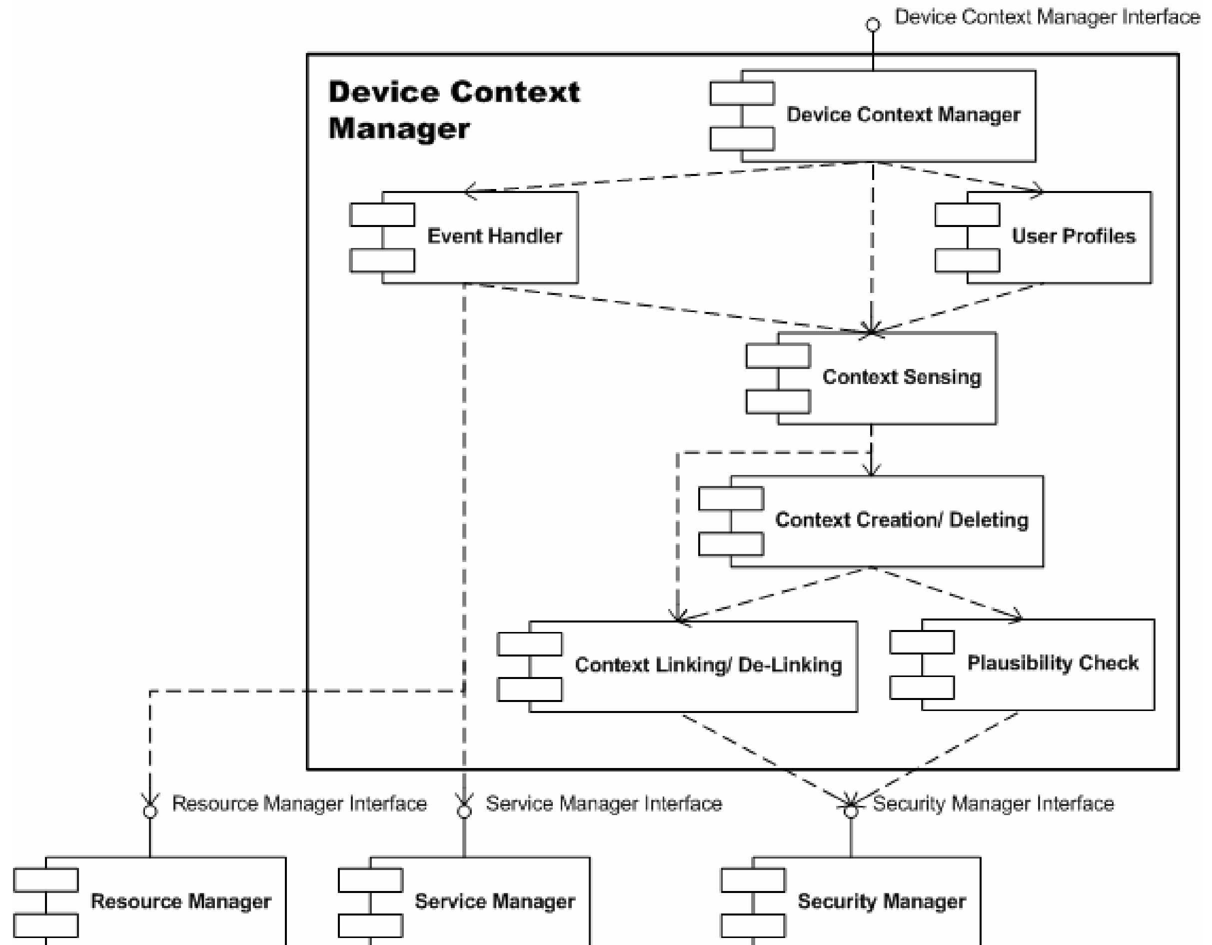

Figure 10 Device Context Manager

Purpose: Responsible for realisation of context for the device elements. This may be on the device itself or on a Hydra-enabled proxy representing the device. It is needed for security and service issues. Additionally it is responsible for event handling.

Main Functions:

- Context Sensing: Gathering data from different sources
- Create and Delete Context
- Linking and De-linking Context
- Manage session based User Profiles
- Check Plausibility of Input
- Event handling: Publishing, subscription

Description: In order to establish a semantic framework as shown in the list below, the Device Context Manager is needed. It realizes the context sensing, the context linking and de-linking and creation and deletion of context.

- Resolution
- Situational Awareness
- Context Awareness
- Context Sensing

On less capable devices the context sensing simply involves status questioning and in some circumstances also location reporting. Most devices will only provide sensor input to derive context. Session based user profiles are used to identify or specify the context. This information is further used by mainly the Device Security and Device Service Manager. Additionally, the Device Context Manager is also responsible for event handling, i.e. event publishing and subscribing. This

information is needed for example by the Device Resource Manager to allocate the resource in an appropriate manner.

Dependencies: Device Service Manager, Device Security Manager, Device Resource Manager

The following requirements are associated with the Device Context Manager:

ID	Description
163	Policy and Context are not part of the middleware
190	Learning situational context
192	Context Modelling
207	Service selection by context
246	The user devices has to be able to establish context

6.2.1.2 Device Device Manager

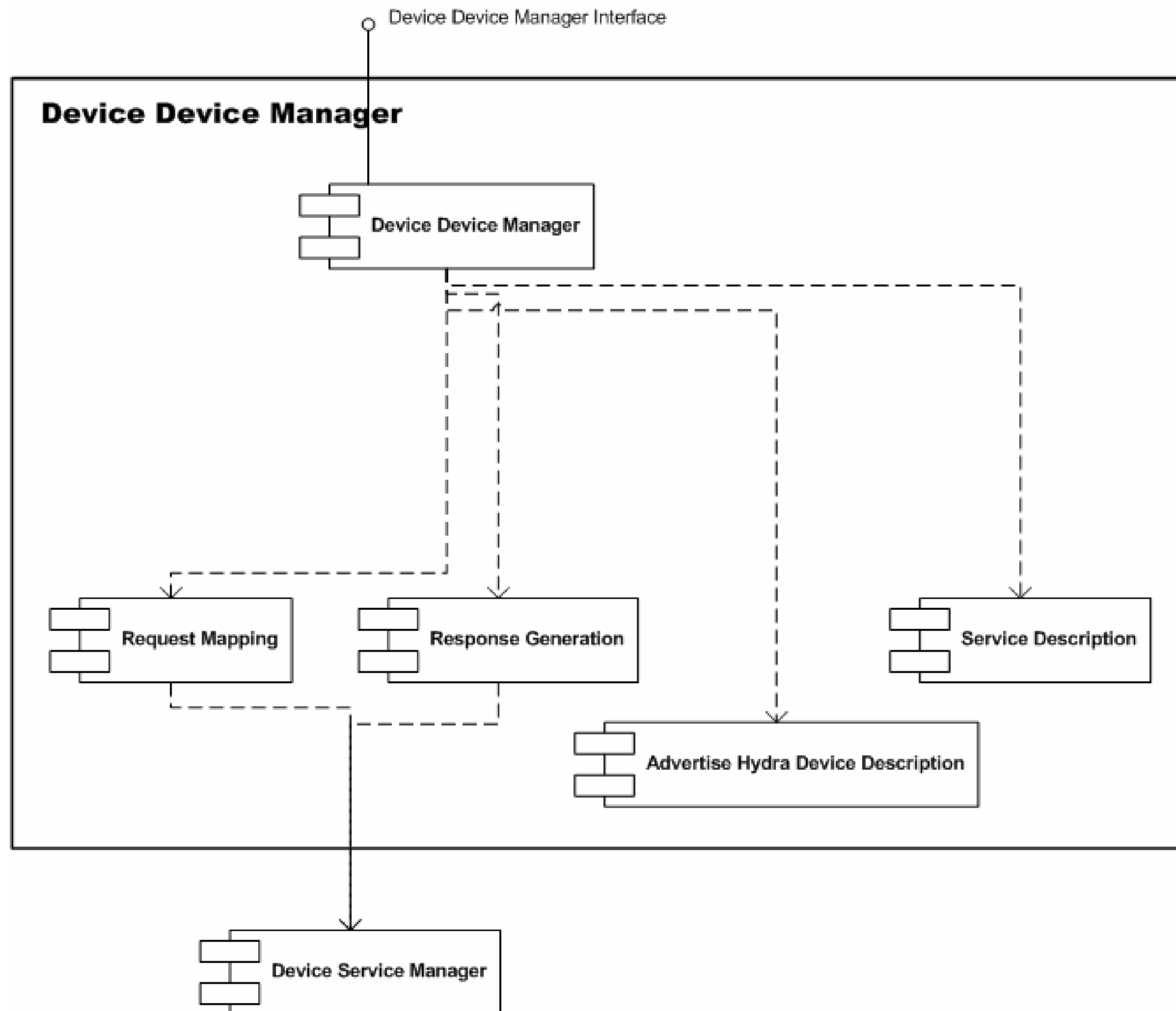

Figure 11 Device Device Manager

Purpose: The Device Device Manager handles several service requests and manages the responses.

Main Functions:

- Maps requests to device services
- Response generation
- Advertising Hydra device description
- Advertises device services

Components:

Advertise: This module is responsible for broadcasting the existence of the device to the outside world. It will support several discovery protocols, at least UPnP (Universal Plug and Play).

Request Mapping: This module maps a request from an outside caller to an internal service in the device.

Response Generator: This module maps translates the result of an internal service in the device to a response to the caller.

Service Description: This module can advertise and provide the service description of the device.

Dependencies: Device Service Manager

The following requirements are associated with the Device Device Manager:

ID	Description
91	Any HYDRA device should have an associated description
92	Rule-based configuration of devices
104	Automatic Discovery of Services
108	Device discovery
109	Device Virtualisation
111	Dynamic Web Service Binding
114	Semantic enabling of device web services
120	Multiple Device Virtualizations
146	Report errors in devices
204	Devices have automatic error diagnostics
239	Automatic service diagnostic for security relevant services
260	Devices Registration
266	Device public services
318	UAAR: Devices should be able to be added to the system at runtime
348	Detect errors in devices
354	Support for virtual devices
376	Security requirements must be part of the HYDRA MDA

6.2.1.3 Device Network Manager

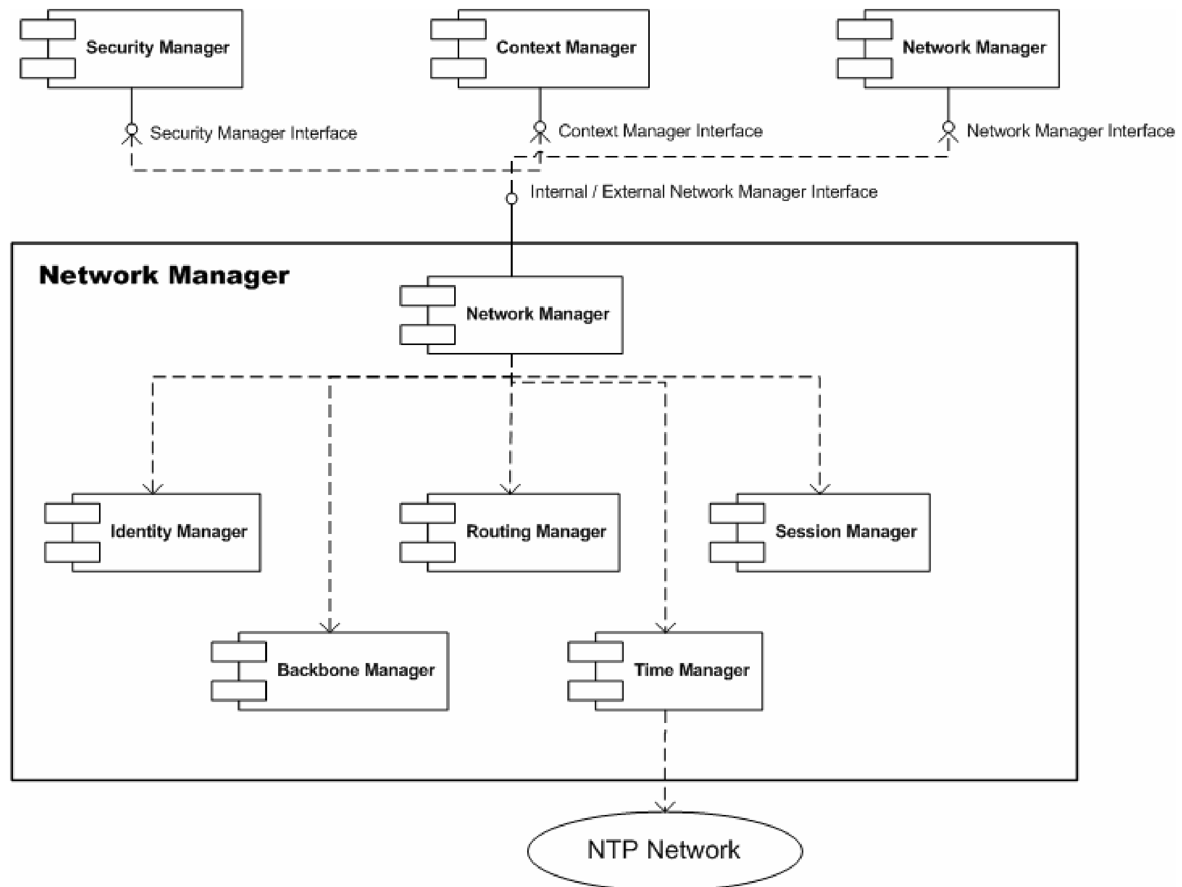

Figure 12 Device Network Manager

Purpose: 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. Network communications are traced with a session mechanism. It also handles the HIDs of the nodes in the application. Finally, it is in charge of synchronizing the nodes in the network with referential time.

The subcomponents in the network manager in the device view are:

Routing Manager: responsible for interacting with the others network entities. It allows sending and receiving messages to/from the network.

Session Manager: responsible for managing sessions within the network communications inside Hydra.

Identity Manager: responsible for managing the Hydra IDs and the correspondences with physical addresses.

Time Manager: responsible for synchronizing the nodes of the application in the network with referential time.

Backbone Manager: Responsible for supporting the Device To Device communication and discovering Hydra IDs over the Hydra network.

Main Functions:

- Unique entry point for the network communications
- Support sessions mechanism
- Synchronise network with referential time
- Provide HID to nodes at application level

- Handle a list of the network members
- Control access to the devices
- Handle information about services in the network

Dependencies: Network Manager (from other deployed middleware), Security Manager and Context Manager

The following requirements are associated with the Device Network Manager:

ID	Description
349	Support for "global time"
404	Identity - Device identity in each context
405	Identity - Device identity management
414	Hydra enabled devices - Single information incoming point and single information outgoing point
415	Backbone - Acting as one unique entity
416	Backbone – Structure
417	Backbone - Distributed information
418	Backbone - Architecture
419	Backbone - Device services and resources announcement through the Gateway
421	Backbone - Ability to reach every device through its device identifier
432	Session Management - Session mechanism inside Hydra
433	Session Management - Persistent sessions
434	Session Management - Validity in time
435	Session Management - Session IDs assignation
436	Session Management - Multiple sessions
437	Session Management - Session IDs uniqueness
454	Identity - Relation between identifiers and physical addresses
455	Identity - Update of the correspondences between identifier and physical addresses
462	Time Manager - Referential time for Hydra Enabled Devices

6.2.1.4 Device Policy Manager

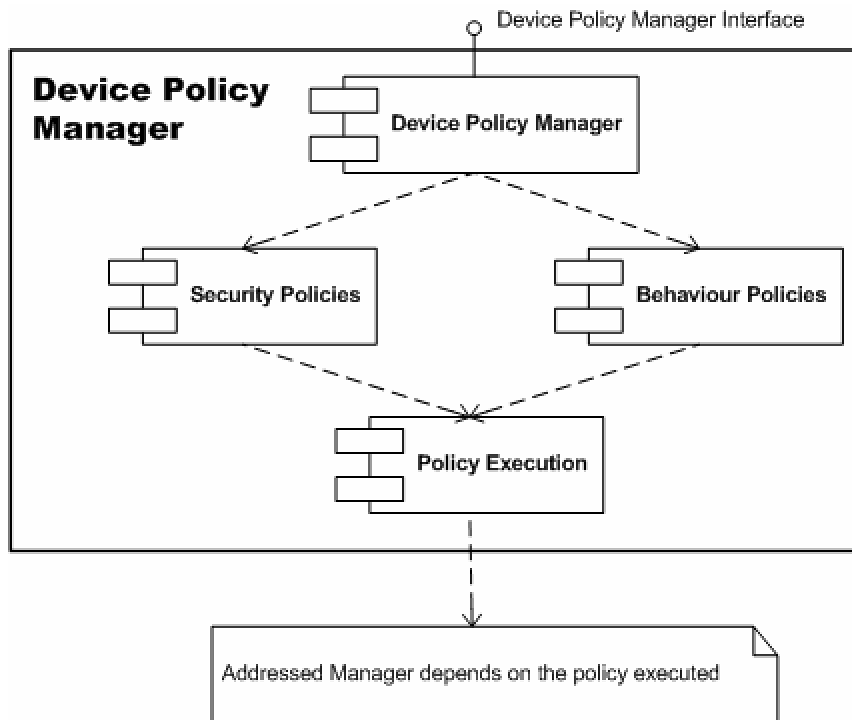

Figure 13 Device Policy Manager

Purpose: Responsible for the execution of the policies. These policies are predefined by the Manufacturer.

Main Functions:

- Execution of Policies

Description: The Device Policy Manager is responsible for execution of predefined policies. These policies are predefined by the manufacturer or developer and can only be updated through a firmware update. The policies can be divided into security and behaviour policies. Latter ones have no direct security implications. The Device Policy Manager invokes the responsible component to execute the policies. After execution, the Device Policy Manager receives a notification about the execution. The Device Policy Manager itself can be invoked by several components, e.g. Device Security Manager, Device Resource Manager or Device Context Manager.

Dependencies: Depends on the executed policies.

The following requirements are associated with the Device Policy Manager:

ID	Description
75	Auto configuration / re-configuration
221	Policy should handle the possible actions
231	Policy-based conflict resolution between services of different clusters
243	The device model has to let end-users control if the devices can be discovered

6.2.1.5 Device Quality of Service Manager

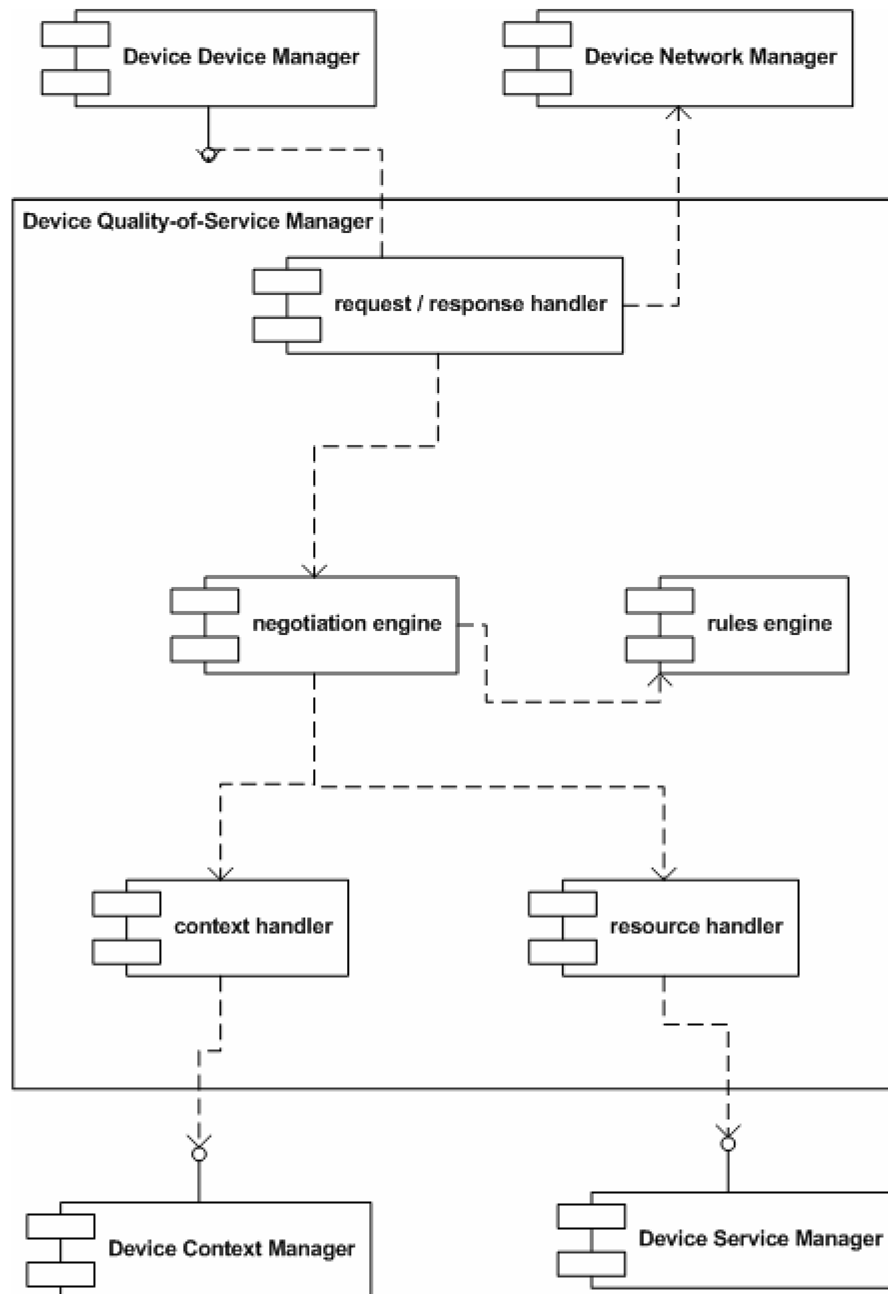

Figure 14 Device Quality of Service Manager

Purpose: The Device QoS Manager is the core component that leads towards a QoS enabled HYDRA middleware. It negotiates QoS parameters with other services and manages resources accordingly. Further, the QoS Manager provides device specific information to the Application Diagnostics Manager.

Main Functions:

- Tracking resource usage,
- Estimate future use of resources,
- Negotiation of service-level agreements,
- Allocation / blocking of resources for exclusive use by certain processes, according to service level agreements,
- Providing information to the Application Diagnostic Manager.

Components:

Request/Response Handler: This module is the interface for client services. It receives requests through the Device Device manager and handles negotiation messages.

Negotiation Engine: This component implements the core of the QoS Manager. Based on information from the rules engine, the context handler and the resource handler, the negotiation engine decides, weather a request can be fulfilled or not. Additionally, it is responsible for allocating/releasing resources.

Rules Engine: The rules engine contains additional information that influence the negotiation engine's decisions. These information could contain device specific restrictions, constraints made by system developers or rules for the implication of contextual information. In contrast to the information provided by the context- and the resource handlers, these information are rather static.

Resource Handler: The resource handler communicates with the Device Resource Manager via the Device Service Manager. It provides information about the device's resources to the negotiation engine.

Context Handler: This component feeds contextual information from the Device Context Manager to the negotiation engine.

Dependencies: Device Service Manager, Device Device Manager, Device Context Manager, Device Resource Manager, Application Diagnostic Manager (through Device Network Manager)

Related requirements:

ID	Description
167	Distributed response composition
174	Coordinated resource sharing
176	Aggregation of resources
177	Dynamic scheduling of resource usage
180	Service mediating network connections according to different qualities
217	The middleware should ensure high robustness of services
291	Quality of Service as search criteria for service selection
330	Communication protocols should support QoS in/between protocols
367	All Effort for QoS
368	Best Effort for QoS

6.2.1.6 Device Resource Manager

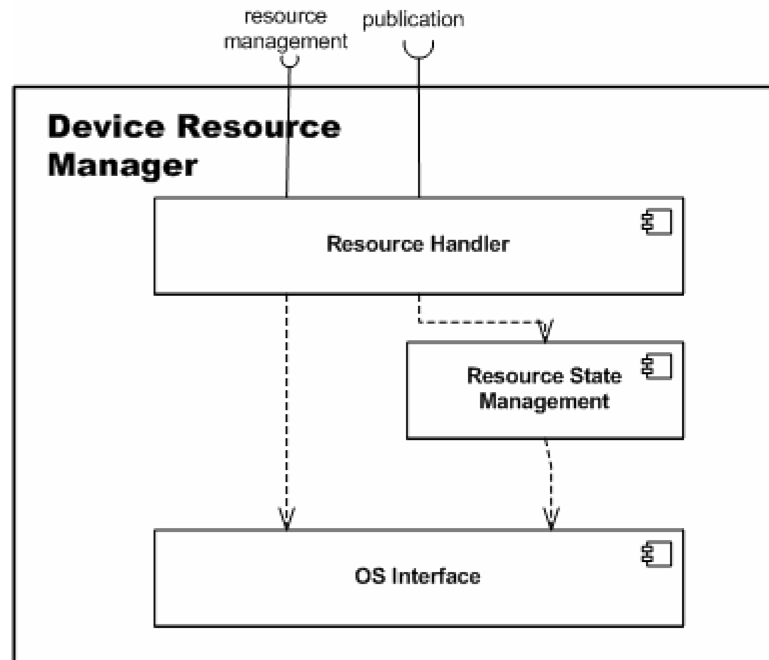

Figure 15 Device Resource Manager

Purpose: Responsible representing and notifying on device and operating system resources.

We distinguish between different levels of resources:

- Base resources that are directly provided by a system. This includes memory, disk space, network bandwidth etc.
- Composite resources that are composed of and use base resources. These include services, clusters, total memory of a device etc.

The resource manager is responsible for tracking base and composite resources on a device.

The OS Interface enables manipulation and query of OS resources. In Java, e.g., it would be subsumed by the Java SDK. On smaller embedded devices, it would have to be created specifically for each OS.

Main Functions:

- Publishing resource data from local device
- Provide an external interface for resource management
- Interfacing to operating system to provide information on local resources.

Dependencies: Device Service Manager, Device Security Manager

The following requirements are associated with the Device Resource Manager:

ID	Description
19	Support of low-end devices
146	Report errors in devices
151	Devices send events when their status changes
233	Self-healing function of middleware
239	Automatic service diagnostic for security relevant services
292	Self-diagnosis of devices
312	UAAR: Support profiling of devices' performance
314	UAAR: Faults should be intercepted by middleware, notified to interested services
348	Detect errors in devices

6.2.1.7 Device Security Manager

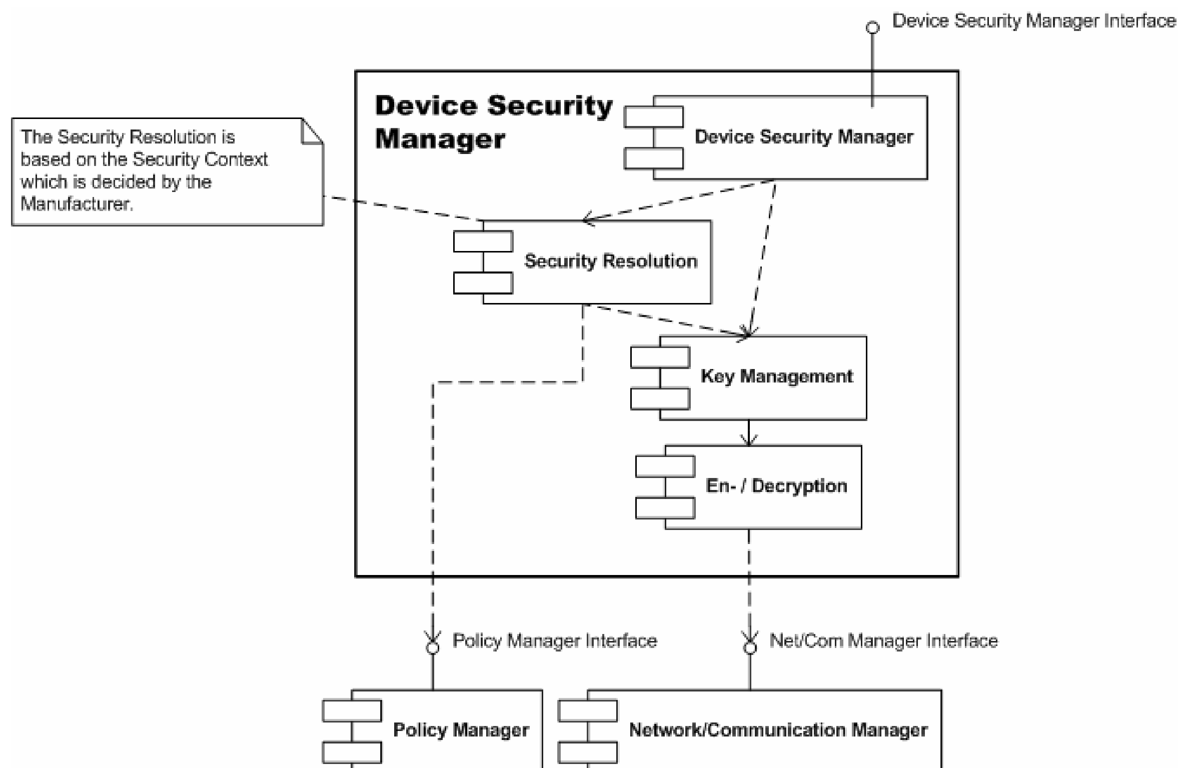

Figure 16 Device Security Manager

Purpose: Realizes the security in the device.

Main Functions:

- Security Resolution based on the security context given by the manufacturer
- Key Management
- Encryption and decryption

Description: The Security Manager is responsible for the security on the devices and the network layer. This includes the encryption and decryption of data sent and received. In order to do that, a key management is provided by the Device Security Manager. Here, keys can be added and removed to ensure that broken keys can be revoked. Also, the Device Security Manager invokes the Device Policy Manager to execute the corresponding security policies, but since the policies are predefined by the manufacturer, the Device Security Manager is not allowed to modify them. Hence, the security resolution is based on the security context defined or decided by the manufacturer or developer.

Dependencies: Device Network Manager, Device Policy Manager

The following requirements are associated with the Device Security Manager:

ID	Description
46	End-user configurability
48	Support for multilateral communication involving several security protocols
50	An identity management must be provided
51	Private communication must be particularly secured
52	A transaction partner should not be able to repudiate the transaction
57	Enable profiling
66	User-centric context-aware access control
70	The security system / model must be highly scalable
74	Building of trust from past experiences
75	Auto configuration/ re-configuration
79	Secure cryptographic key management
88	The system should be interoperable with biometric authentication supporting revocable keys
142	Knowledge model for security
178	Single sign-on, run anywhere
249	HYDRA should be open to indirect authentication
296	Adaptability of Security Model with regard to existing security system(s)
297	Secure data erasing on HYDRA enabled devices
301	Most appropriate security model(s) should be proposed by devoloper
302	Security Support for IPv6
308	The security level of an existing network should be determinable
358	Developer must be able to semantically define security requirements
363	The security model should support revocable keys
374	Each application must be able to have its own security model
446	It must be possible to negotiate different (security) parameters while establishing a connection

456	The middleware must support different security mechanisms as required by national laws
468	Different levels of security must be supported

6.2.1.8 Device Service Manager

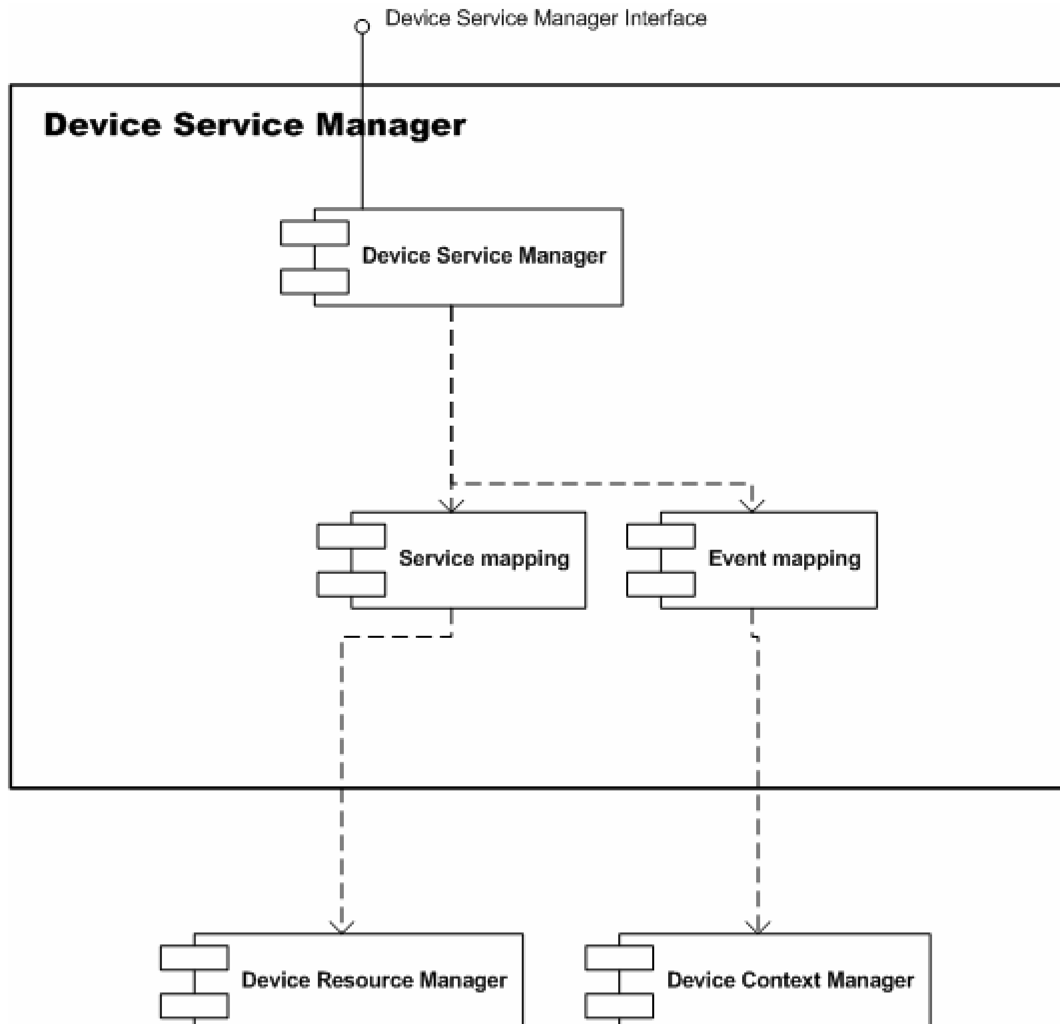

Figure 17 Device Service Manager

Purpose: Implements service interface for physical devices.

Main Functions:

- Maps services to physical device operations
- Maps (physical) device events to Hydra enabled events

Components:

Service Mapping: This module maps device service request to internal device operations. One device can have several service mappings.

Event Mapping: This module physical device events and maps them into Hydra-events.

Dependencies: Device Resource Manager, Device Context Manager

The following requirements are associated with the Device Service Manager:

ID	Description
17	When applicable, middleware interfaces are exposed by WSA-compatible services
120	Multiple Device Virtualizations
148	Access to basic and extended functionality
153	Automatic generation of user interface
159	Service brokers must be organized in a hierarchical way
164	Support for Service standards
206	Middleware supports service discovery
329	Middleware provides domain-independent services
376	Security requirements must be part of the HYDRA MDA

6.2.1.9 Device Storage Manager

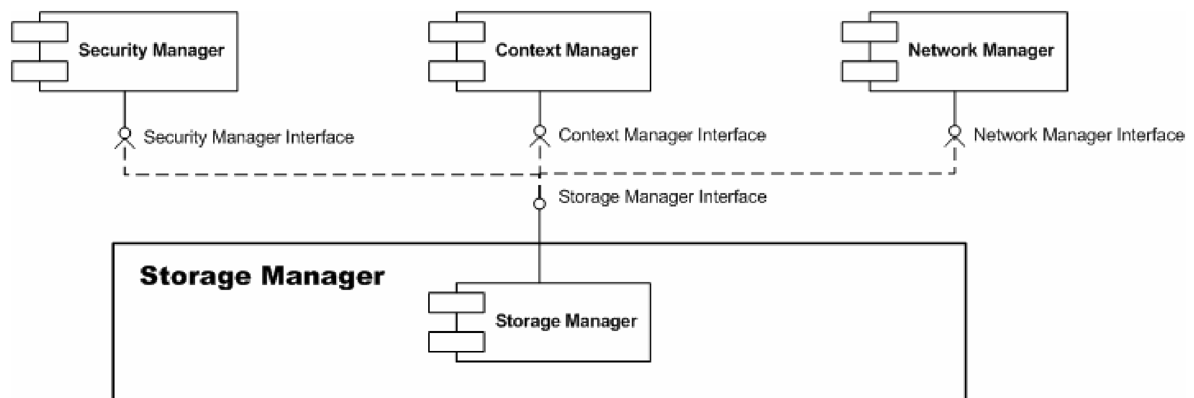

Figure 18 Device Service Manager

Purpose: The Storage Manager is responsible for storage management on the device level. It is needed to provide a single, consistent interface to all storage on the device. Depending on the device capabilities, the Storage Manager may handle the actual storage locally or take care of handling it through remote devices. Typical storage requirements include file and memory storage. Limited data sets to be used frequently and possibly by several cooperating devices, can be stored in a virtual shared memory structure to allow efficient and easy application programming, whereas big or less frequently used data sets are more suitable for file storage. In both cases it is the responsibility of the Storage Manager to provide a familiar interface to the storage and take care of the actual underlying file and memory operations.

Main Functions:

- Unique entry point for memory and file storage
- Control access to storage
- Handle mapping of virtual memory and files to actual local or remote storage
- Support synchronized (locked) access to storage

Dependencies: Network Manager (for remote Storage Manager access), Security Manager and Context Manager

The following requirements are associated with the Device Storage manager:

ID	Description
406	Storage Manager - Gateways information gathered storage
407	Storage Manager - Gateways information stored synchronization
409	Storage Manager - Device information metadata
443	Storage Manager - Gateways must allow efficient access to store data from associated devices

6.2.2 Application Elements

The following diagram explains how the application elements are logically grouped and in the following chapters each element will be explained in detail:

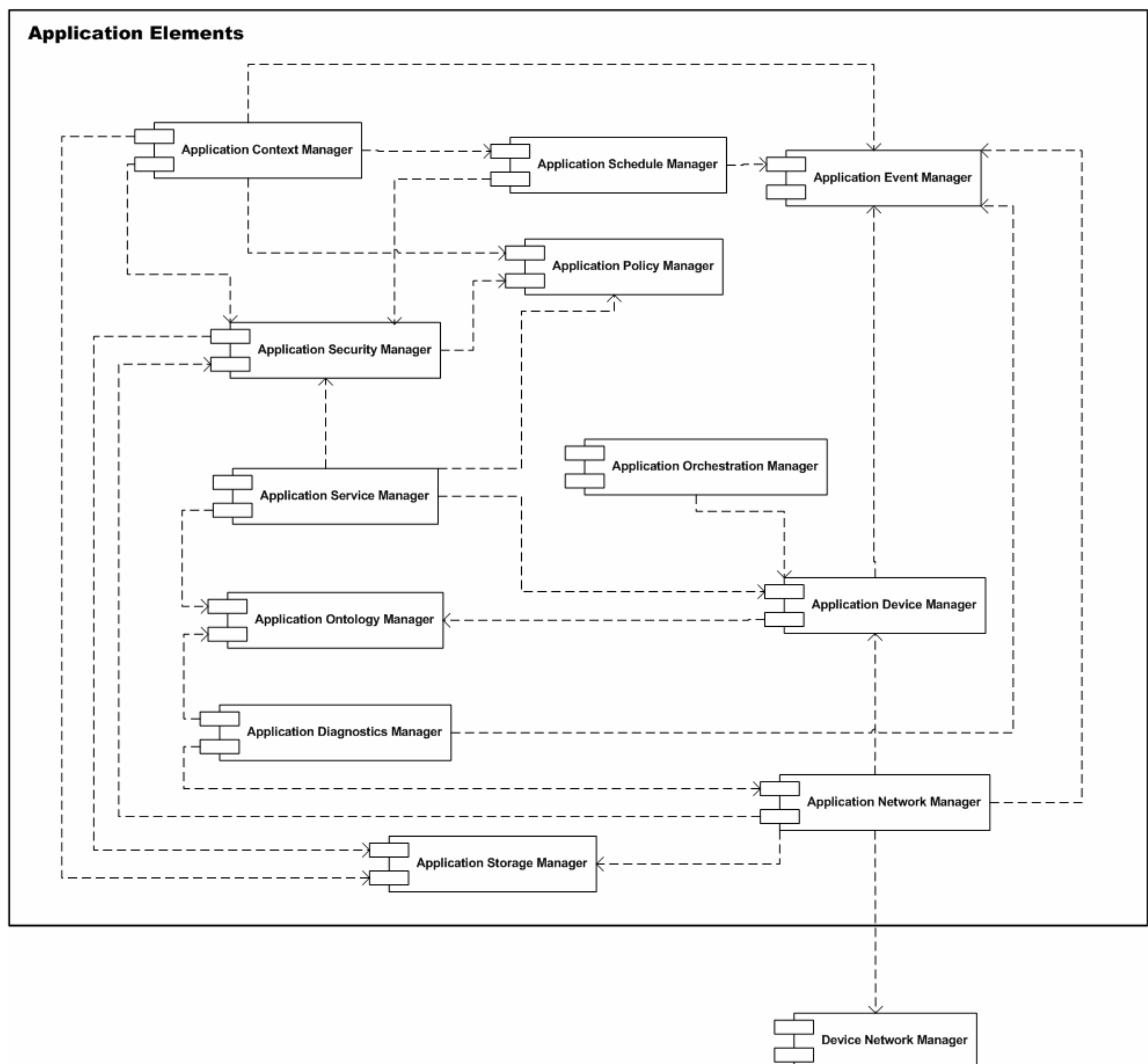

Figure 19 Overview of Application Elements

6.2.2.1 Application Context Manager

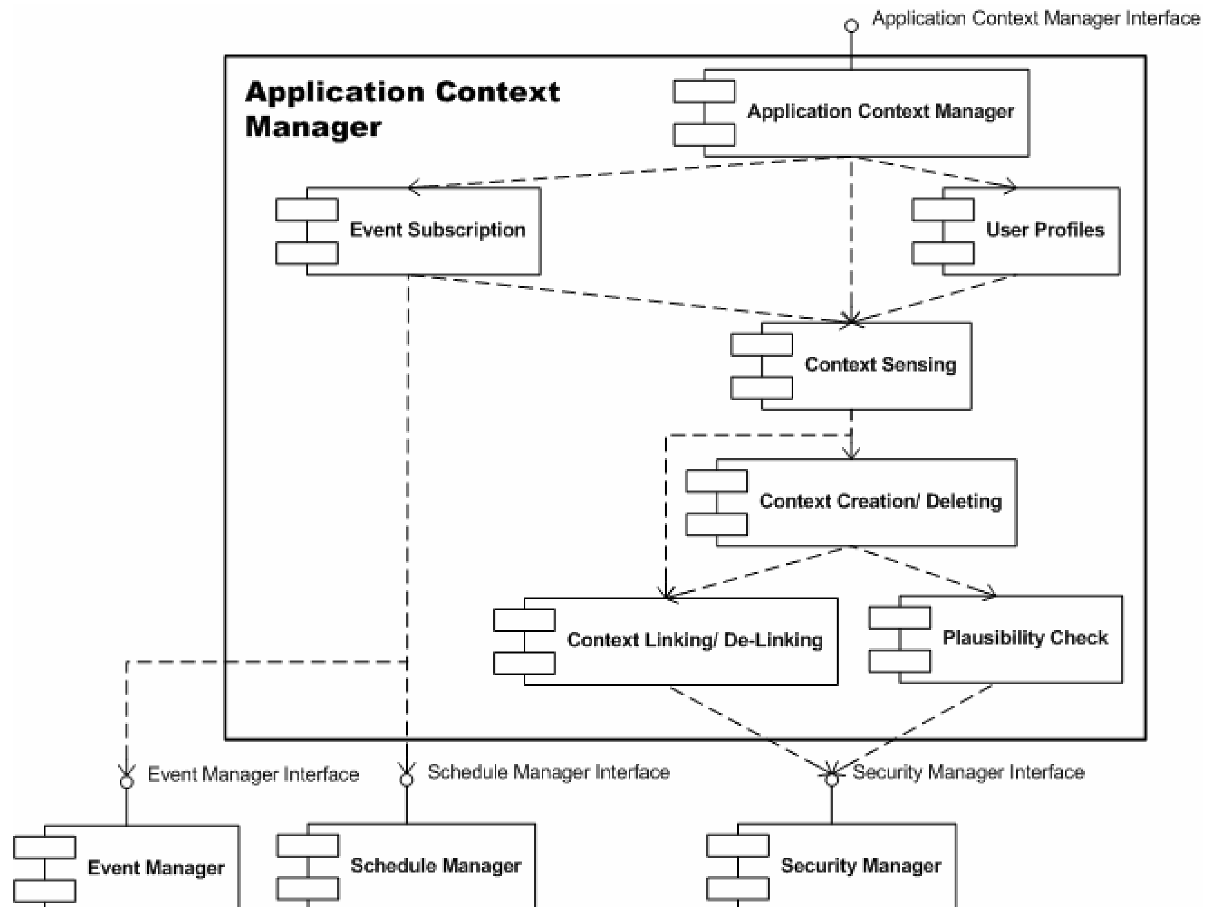

Figure 20 Application Context Manager

Purpose: Responsible for realization of context for the application elements.

Main Functions:

- Context Sensing through data gathering from different sources
- Context Creation and Deleting
- Context Linking and De-linking
- Plausibility check
- Management of User Profiles

Description: In order to establish a semantic framework as shown in Figure 20, the Application Context Manager is needed. It realizes the context sensing, the context linking and de-linking and creation and deletion of context.

- Resolution
- Situational Awareness
- Context Awareness
- Context Sensing

User Profiles are stored only on user device such as PDA's, but application based profiles can be stored for short time to enable applications to execute user specified tasks. The established context can be used in several ways. This includes the security, which is thus based on the whole information available and not only on parts. Also, the context can be used by the Application Service and Application Device Manager in order to identify the appropriate services and devices to execute a task. Since there is a plausibility check of the input data the Application Context Manager also interacts with the Application Diagnostics Manager to ensure that no faulty input is used for application execution.

Dependencies: Application Schedule Manager, Application Event Manager, Application Security Manager, but also Application Device Manager, Application Service Manager, Application Diagnostics Manager

The following requirements are associated with the Application Context Manager:

ID	Description
163	Policy and Context are not part of the middleware
190	Learning situational context
192	Context Modelling
207	Service selection by context
246	The user devices has to be able to establish context

6.2.2.2 Application Device Manager

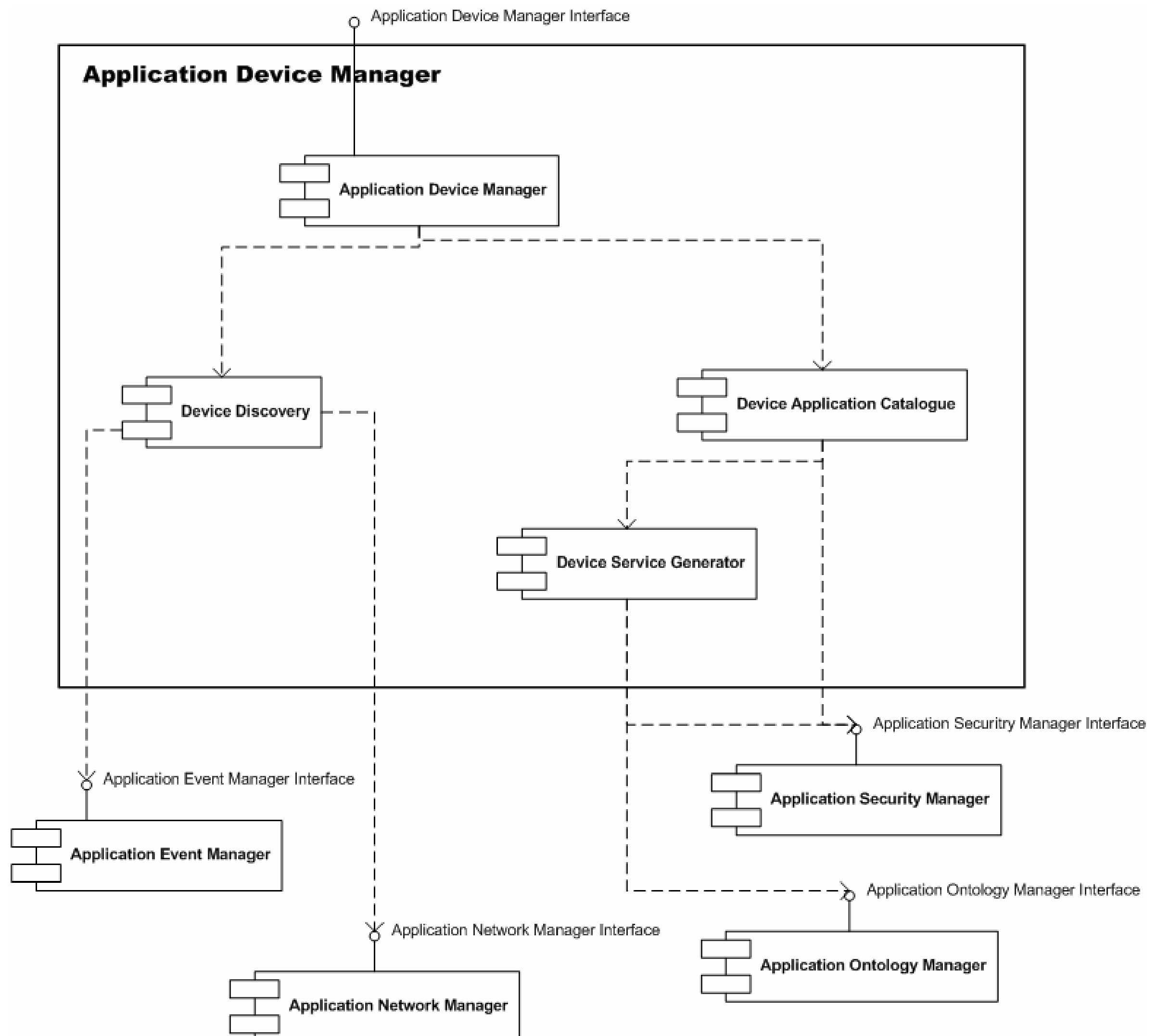

Figure 21 Application Device Manager

Purpose: The Application Device Manager manages all knowledge and information regarding devices. The Application Device manager knows about devices from a network perspective but do not handle the locations or context of the devices.

Main Functions:

- Discover new (existing) devices
- Resolves device communication requests
- Returns service interface
- Returns device communication end-points
- Handles device aggregation
- Handles device virtualization
- Manages a Device Application Catalogue: Device types and device states

Components: There are three main components of the Device Manager.

Discovery Module: One of the major functions of the Application Device Manager is to discover new devices in the network. This is taken care of by the Discovery Module. It will support UPnP discovery (Universal Plug and Play). It will also support user-initiated discovery schemes. Requirements 108 and 218 are associated with this module.

Device Application Catalogue: The Device Application Catalogue keeps track of and manages all devices that are currently active within one application. It can be queried about existing devices and their status. It can also provide service interfaces for the different devices upon request. The Device Application Catalogue will also keep track of when the device entered the system, when it was last heard of and its current "error state". The "error state" will reflect if the Device Application Catalogue believes that the device is working. This state should be maintained by the Application Diagnostic Manager. The Device Application Catalogue should also provide methods for removing devices, i.e. that devices that are removed can unregister themselves from the catalogue. Requirements 91, 98, 110 and 111 are associated with this module.

Device Service Generator: The Device Service Generator is responsible for generating a service interface for a certain device. It will create a software wrapper around the device which other modules can use to communicate with and control the device. Requirements 91, 111, 120 and 325 are associated with this module.

Dependencies: Application Ontology Manager, Application Event Manager, Application Network Manager and Application Security Manager

The following requirements are associated with the Application Device Manager:

ID	Description
14	Automatic device discovery
91	Any HYDRA device should have an associated description
98	Detection of device failures
108	Device discovery
109	Device Virtualisation
110	Device Categorization
111	Dynamic Web Service Binding
112	Dynamic Web Service Generation
113	Composition (of services and devices)

120	Multiple Device Virtualizations
160	Search masks for device/service discovery
179	Dynamic resource handling
189	Plug and play support for adding devices
209	Middleware has a service for providing information about the technical environment/infrastructure
218	Support interaction devices
247	Integrate non-HYDRA devices with an existing HYDRA environment
260	Devices Registration
262	Store information about the attached devices in a central element
271	Remove devices
288	Query devices for their functionality
325	Support aggregation and separation of devices and services
372	Interfacing with external systems
376	Security requirements must be part of the HYDRA MDA

6.2.2.3 Application Diagnostics Manager

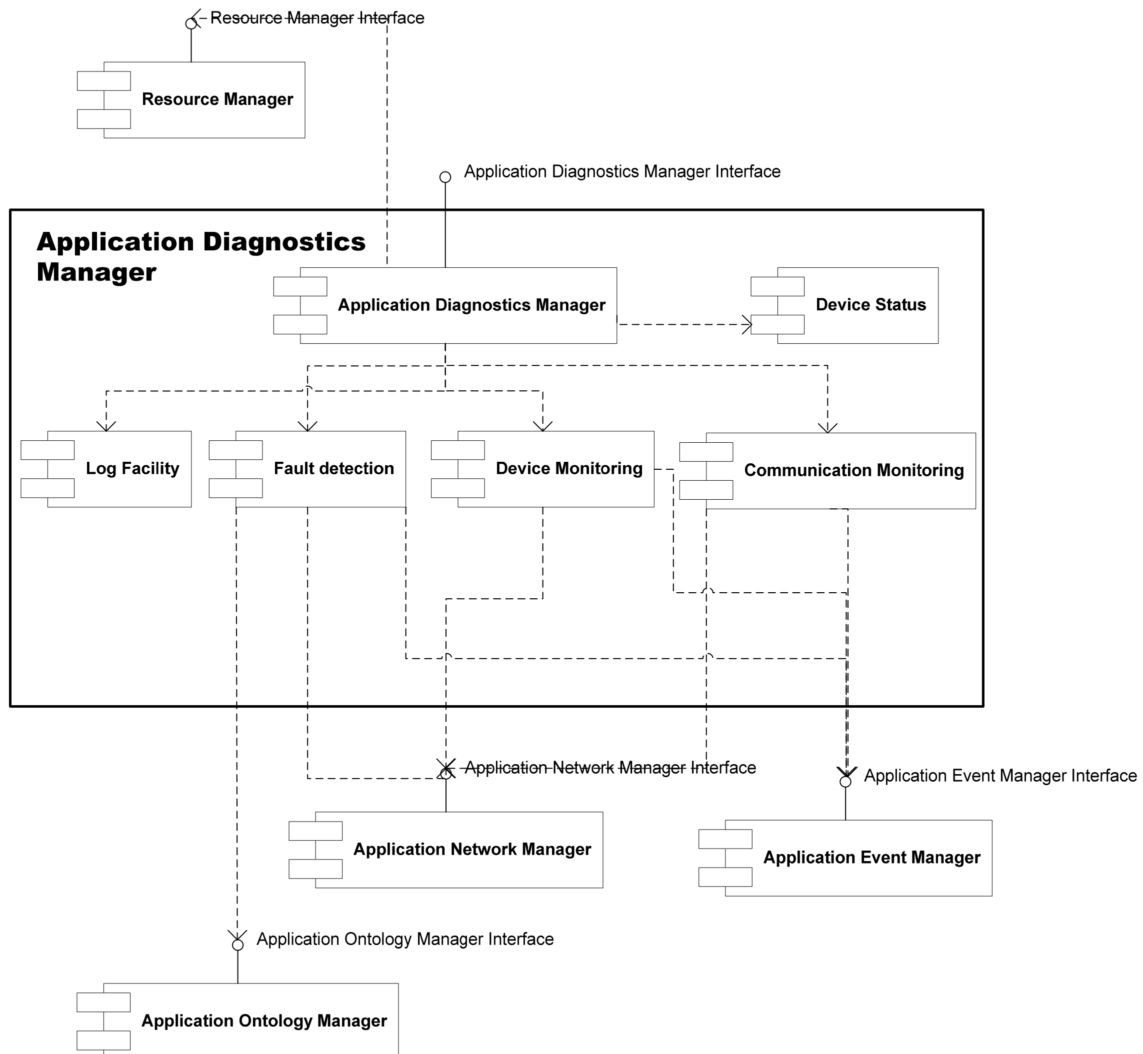

Figure 22 Application Diagnostics Manager

Purpose: The purpose of the Application Diagnostics Manager is to monitor the system conditions and state. It will be responsible for error detection and logging of device events. The Diagnostics Manager will be an important component in providing the self-* properties of Hydra. Completely reliable failure detection is impossible in a distributed system with the characteristics of Hydra, so the Diagnostics Manager will need to work with imperfect failure detectors.

Main Functions:

- Systems diagnostics (e.g., a device is dead/ doesn't respond)
  - dead/live lock detection
  - software failure
  - hardware failures
  - network failures
- Device Diagnostics (device responds but…)
  - service failure
  - device status reports
- Application diagnostics / Monitoring
  - global resource consumption
  - overall property use (e.g., room is too warm)

- Logging
- Fault detection rule engine
    - manages rules/dependencies over devices

Components:

Device Status: The Device Status module is responsible for finding out the status of a device and if there are any malfunctions detected. This component should be coordinated with the device state machine running on the Resource Manager component in order to get all the information of interest.

Log Facility: The Log Facility is used to log all events and interactions between devices. This is used by several other modules to implement their functionality. The log can also be used to detect different erroneous states.

Fault Detection: This component will execute rules or rule sets to discover if there is any malfunctioning or strange behavior in the system. Recovery actions can also be published or taken in order to achieve self-managing.

Device Monitoring: This component is used to process rules or rule sets to monitor devices in order to be preemptive to avoid errors and malfunctions, for instance by monitoring the resource usage of certain devices.

Communication Monitoring: This component is used to conduct packet sniffing on the host running the Web Services and then can be used to make decisions on the working status of the device.

Dependencies: Application Ontology Manager, Application Event Manager, Application Network Manager

The following requirements are associated with the Application Diagnostics Manager:

ID	Description
91	Any HYDRA device should have an associated description
94	Simulation environment
96	Detect deadlocks
97	Detect livelocks
98	Detection of device failures
101	Manual device ontology definition
103	Automatic device ontology construction
104	Automatic Discovery of Service
108	Device discovery
110	Device Categorisation
117	HYDRA component ontology
119	Domain modelling support
122	Configurable and easy to install middleware

123	Support updates at run-time
125	Transactional updates
126	Automatic Device ontology updates
139	Knowledge model of HYDRA middleware
141	Download and harmonisation of third party device ontologies
164	Support for services standards
226	Device ontology should be available
243	The device model has to support the ability of the end-user control if the device can be discovered
262	Store information about the attached devices in a central element
359	Device ontology versioning
365	Ability to self-adaptation
371	Devices classes hierarchy
376	Security requirements must be part of the HYDRA MDA
392	Rules for selection of alternative devices

6.2.2.4 Application Event Manager

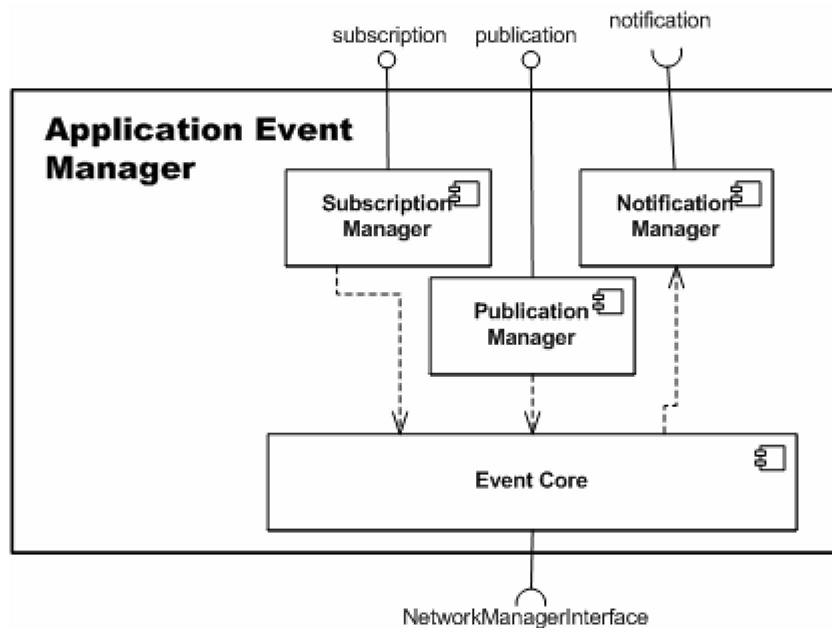

Figure 23 Application Event Manager

Purpose: Responsible for providing publish/subscribe functionality to the HYDRA middleware.

In general, publish/subscribe communication as provided by the Event Manager 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 Application Event Manager will be use in any place where there is a potential many-to-many relationship between senders and receivers and where asynchronous communication is desirable.

Main Functions:

- Subscription support allowing clients to subscribe to published events via a topic-based publish/subscribe scheme (Subscription Manager)
- Publication support allowing client to publish event on topics (Publication Manager)
- Routing events to subscribed clients (Notification Manager)
- Event Core manages persistent subscriptions, publication to subscription matching etc.
- Interfacing to Network Manager (e.g., broadcast-, multicast-, or gossiping-based dissemination)

Dependencies: Application Security Manager, Application Network Manager

The following requirements are associated with the Application Event Manager:

ID	Description
93	Re-playable event logging
151	Devices send events when their status changes
153	Automatic generation of user interface

6.2.2.5 Application Network Manager

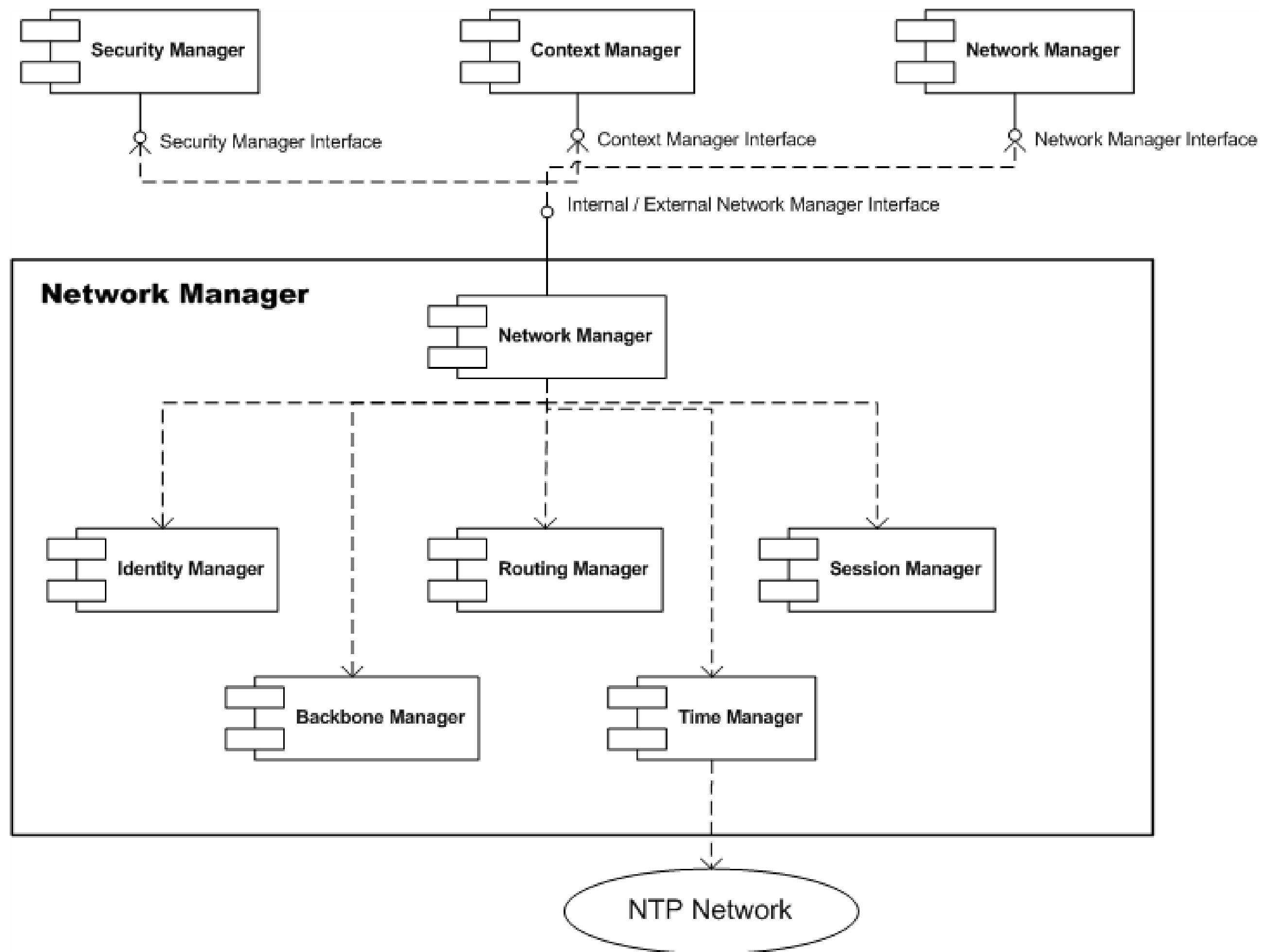

Figure 24 Network Manager

Purpose: 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. Network communications are traced with a session mechanism. It also handles the HIDs of the nodes in the application. Finally, it is in charge of synchronizing the nodes in the network with referential time.

The subcomponents in the network manager in the device view are:

Routing Manager: responsible for interacting with the others network entities. It allows sending and receiving messages to/from the network.

Session Manager: responsible for managing sessions within the network communications inside Hydra.

Identity Manager: responsible for managing the Hydra IDs and the correspondences with physical addresses.

Time Manager: responsible for synchronizing the nodes of the application in the network with referential time.

Backbone Manager: Responsible for supporting the Device To Device communication and discovering Hydra IDs over the Hydra network.

Main Functions:

- Unique entry point for the network communications
- Support sessions mechanism

- Synchronise network with referential time
- Provide HID to nodes at application level
- Handle a list of the network members
- Control access to the devices
- Handle information about services in the network

Dependencies: Network Manager (from other deployed middleware), Security Manager and Context Manager

The following requirements are associated with the Application Network Manager:

ID	Description
349	Support for "global time"
404	Identity - Device identity in each context
405	Identity - Device identity management
414	Hydra enabled devices - Single information incoming point and single information outgoing point
415	Backbone - Acting as one unique entity
416	Backbone – Structure
417	Backbone - Distributed information
418	Backbone - Architecture
419	Backbone - Device services and resources announcement through the Gateway
421	Backbone - Ability to reach every device through its device identifier
432	Session Management - Session mechanism inside Hydra
433	Session Management - Persistent sessions
434	Session Management - Validity in time
435	Session Management - Session IDs assignation
436	Session Management - Multiple sessions
437	Session Management - Session IDs uniqueness
454	Identity - Relation between identifiers and physical addresses
455	Identity - Update of the correspondences between identifier and physical addresses
462	Time Manager - Referential time for Hydra Enabled Devices

6.2.2.6 Application Ontology Manager

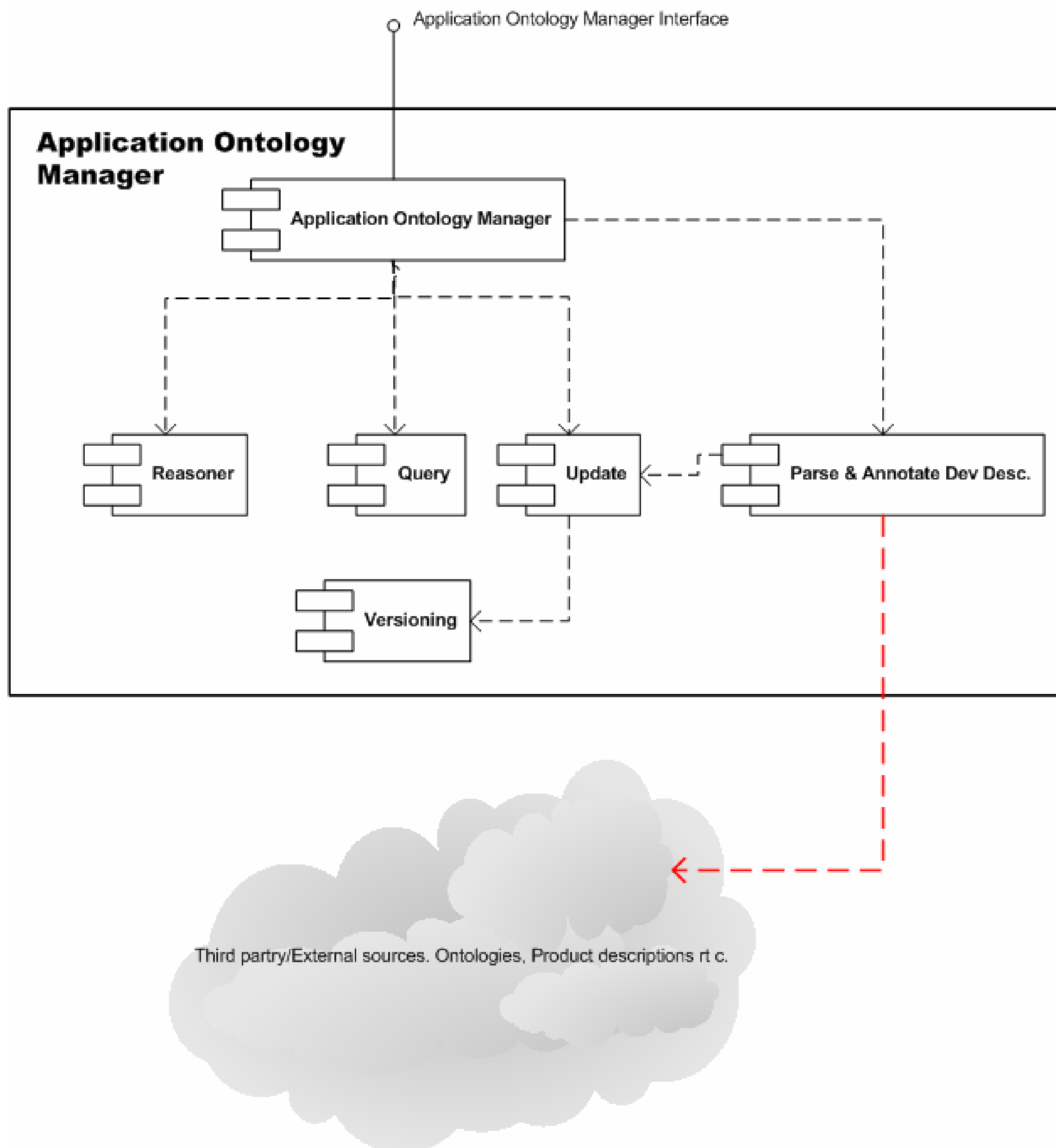

Figure 25 Application Ontology Manager

Purpose: One of the key components in the Hydra middleware is the Device Ontology, where all meta-information and knowledge about devices and device types are stored. The purpose of the Application Ontology Manager is to provide an interface for using the Device Ontology. This manager could possibly also maintain other models in addition to devices.

Main Functions:

- Device description & annotation
- Parsing & annotation of device description
- Search/Query function
- Update
- Ontology versioning
- Reasoning module

Components:

Reasoner: The reasoner module is responsible for reasoning about devices and their status and provides inferencing mechanisms for instance to conclude what type of device has entered the network.

Query module: The query module allows for retrieving information regarding devices and their capabilities.

Update module: The update module allows entering of new information, deletion and changes to the ontology at both design time and run time.

Versioning: The versioning module is responsible for managing different version of the ontology. This includes different versions of devices and services.

Parse & Annotate: The parse & annotate modules is responsible for automatically update the ontology with new device types. It does so by analyzing and annotate existing device and product descriptions which is fed into the ontology.

Dependencies: External ontologies and product description databases

The following requirements are associated with the Application Ontology Manager:

ID	Description
91	Any HYDRA device should have an associated description
98	Detection of device failures
101	Manual device ontology definition
103	Automatic device ontology construction
104	Automatic Discovery of Services
108	Device discovery
110	Device Categorization
117	Hydra component ontology
119	Domain modelling support
123	Support updates at run-time
125	Transactional updates
126	Automatic Device ontology updates
139	Knowledge model of Hydra middleware
141	Download and harmonisation of third party device ontologies
164	Support for Service standards
226	Device ontology should be available
243	The Device Model has to support the ability of the End-user to control if the device can be discovered
262	Store information about the attached devices in a central element
359	Device ontology versioning

365	Ability to self-adaptation
371	Devices classes hierarchy
376	Service requirements must be part of the Hydra MDA
392	Rules for selection of alternative devices

6.2.2.7 Application Orchestration Manager

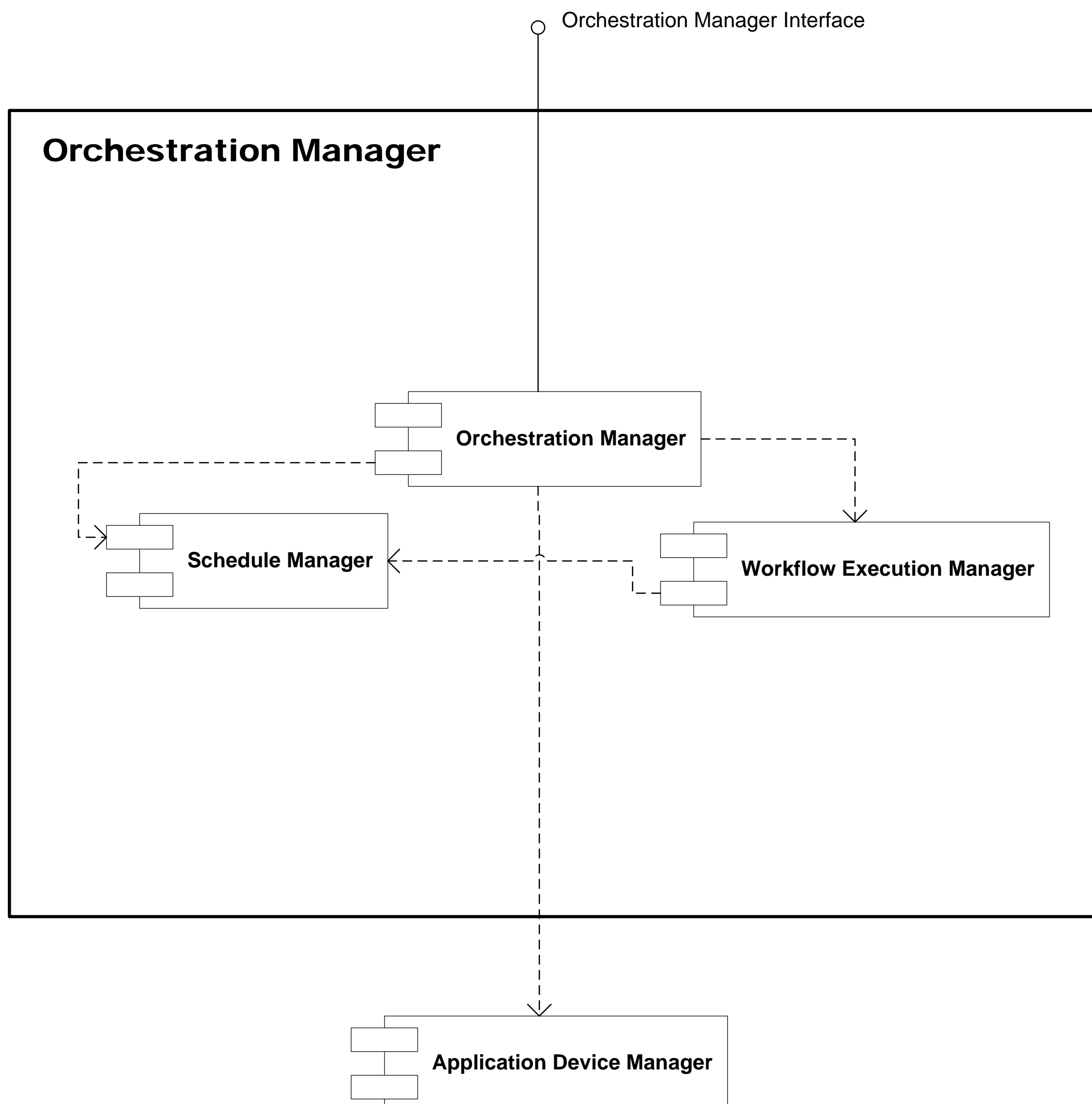

Figure 26 Application Orchestration Manager

Purpose: The Application Orchestration Manager provides support for composite services and workflows. It is an execution engine for the Hydra Device Orchestration Language ("DOLL").

Main Functions:

- Execute call sequences consisting of invocations of Device services
- Provide scheduling of notifications and service calls for Hydra applications

Components:

Schedule Manager: The scheduler is responsible for running tasks or notifying applications when a specific criteria is met. Such a criteria can be a e.g. specific (possibly recurring) time, system startup, system shutdown.

Workflow Execution Manager: The workflow execution module interprets process descriptions and executes a set of services. These processes may represent a complex service composed of other services or part of a HYDRA application.

Dependencies: Application Device Manager

The following requirements are associated with the Application Orchestration Manager:

ID	Description
113	Composition (of services and devices)
392	Rules for selection of alternative devices
376	Security requirements must be part of the Hydra MDA

6.2.2.8 Application Policy Manager

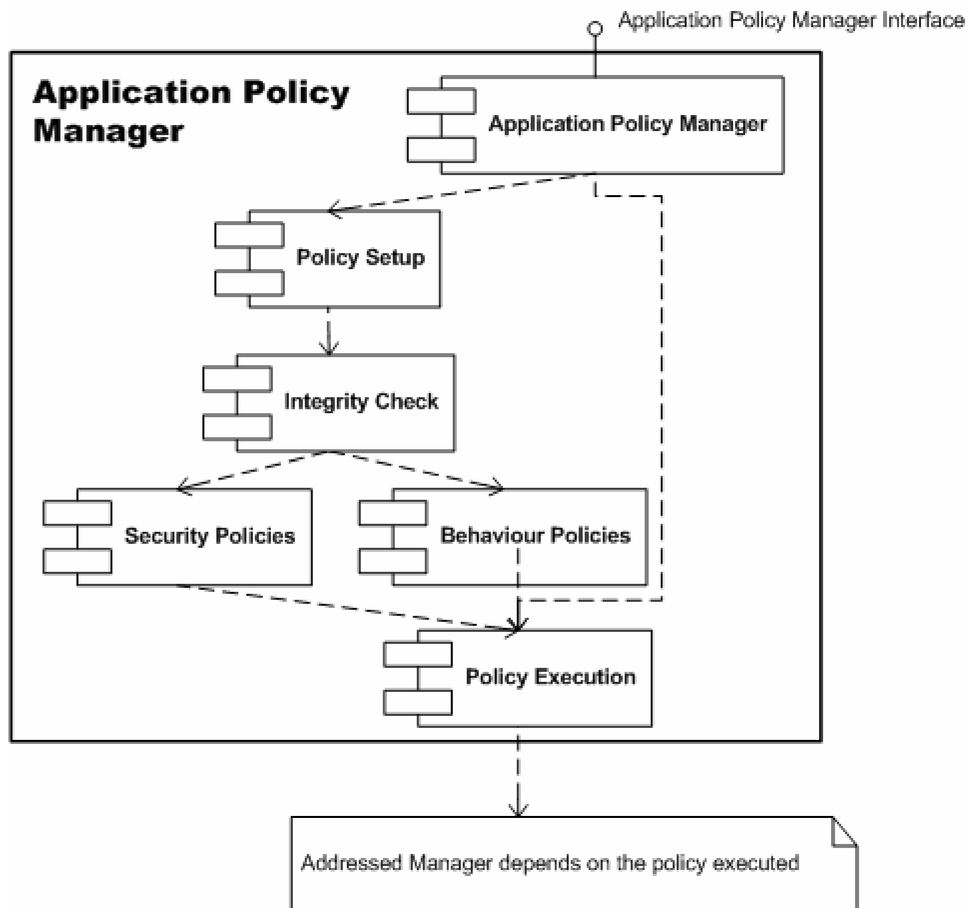

Figure 27 Application Policy Manager

Purpose: Policy Management and execution in the application.

Main Functions:

- Policy setup based on definitions from the Application Security Manager
- Integrity Check to ensure that new policies do not interfere with existing policies
- Policy Execution

Description: The Policy Manager handles the policies which are divided into security and behaviour policies. The security policies can be updated by the Application Security Manager. Thus, an integrity check is performed before new rules are added or old are removed to ensure that all rules are conforming and that one rule is not contradicting to another rule. For execution of the rules the Application Policy Manager invokes the responsible manager. Therefore, several managers are depending on the Application Policy Manager, such as the Application Service Manager, the Application Device Manager or the Application Event Manager. After execution the policy manager receives a notification.

Dependencies: Depends on the executed policies.

The following requirements are associated with the Application Policy Manager:

ID	Description
75	Auto configuration / re-configuration
221	Policy should handle the possible actions
231	Policy-based conflict resolution between services of different clusters
243	The device model has to let end-users control if the devices can be discovered

6.2.2.9 Application Schedule Manager

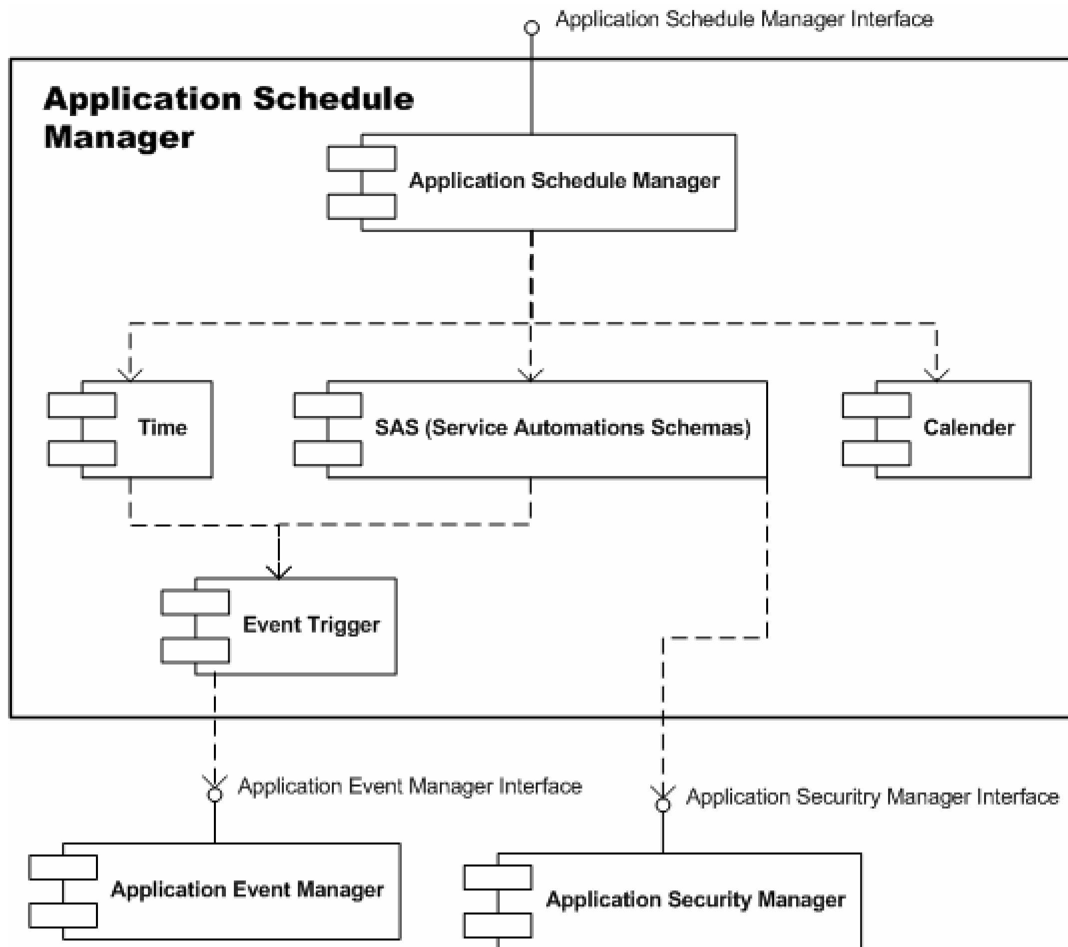

Figure 28 Application Schedule Manager

Purpose: Implements service automation schemes (SAS). Performs event scheduling (including trigger functionality).

Main Functions:

- Event triggering
- Calendar function
- Triggering
- Device coordination
- Schedule execution (based on service automation schemes)

Dependencies: Application Event Manager, Application Security Manager

The following requirements are associated with the Application Scheduling Manager:

ID	Description
177	Dynamic scheduling of resource usage

6.2.2.10 Application Security Manager

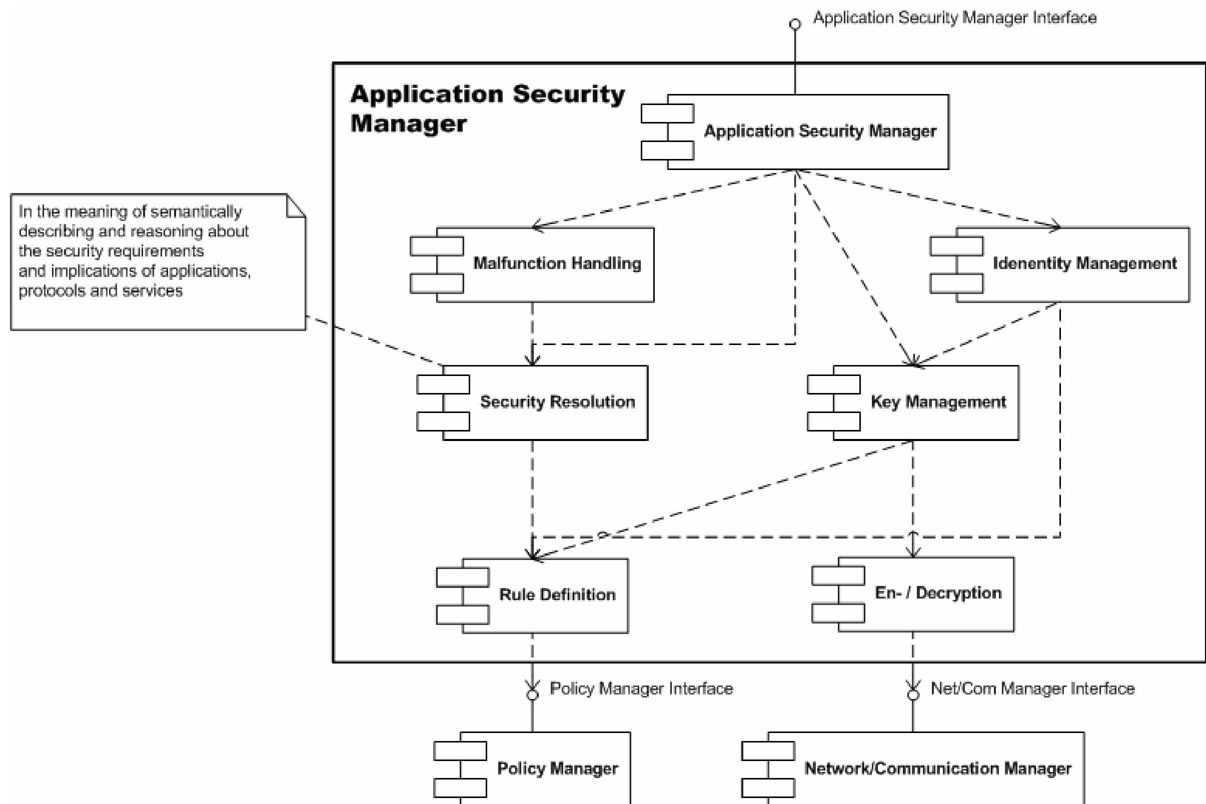

Figure 29 Application Security Manager

Purpose: Realizes the security aspects.

Main Functions:

- Malfunction Handling based on the reasoning/plausibility check of the Application Context Manager
- Security Resolution based on the application, device and service requirements
- Identity Management
- Key Management
- Rule definition for the Application Policy Manager
- Encryption and decryption on the application level

Description:

The Application Security Manager is responsible for the security on powerful devices and in the application view. It has a malfunction handling based on the plausibility check of the Application Context Manager and the results of the Application Diagnostics Manager. Identity management is provided in order to enable a solid key management and rules definition. Of course, the Application Security Manager is also responsible for en- and decryption of application and user specific data, regardless of the further encryption on the device view. Out of the semantic security resolution based on the decision of the end-user, application device and service requirements the Security Manager defines rules which are passed to the Application Policy Manager to update its policy set. The input of the resolution originates from various managers such as the Application Device Manager, the Application Service Manager or the Application Context Manager.

Dependencies: Application Policy Manager, Application Network Manager

The following requirements are associated with the Application Security Manager:

ID	Description
46	End-user configurability
48	Support for multilateral communication involving several security protocols
50	An identity management must be provided
51	Private communication must be particularly secured
52	A transaction partner should not be able to repudiate the transaction
57	Enable profiling
66	User-centric context-aware access control
70	The security system / model must be highly scalable
74	Building of trust from past experiences
75	Auto configuration/ re-configuration
79	Secure cryptographic key management
88	The system should be interoperable with biometric authentication supporting revocable keys
142	Knowledge model for security
178	Single sign-on, run anywhere
249	HYDRA should be open to indirect authentication
296	Adaptability of Security Model with regard to existing security system(s)
297	Secure data erasing on HYDRA enabled devices
301	Most appropriate security model(s) should be proposed by devoloper
302	Security Support for IPv6
308	The security level of an existing network should be determinable
358	Developer must be able to semantically define security requirements
363	The security model should support revocable keys
374	Each application must be able to have its own security model
446	It must be possible to negotiate different (security) parameters while establishing a connection
456	The middleware must support different security mechanisms as required by national laws
468	Different levels of security must be supported

6.2.2.11 Application Service Manager

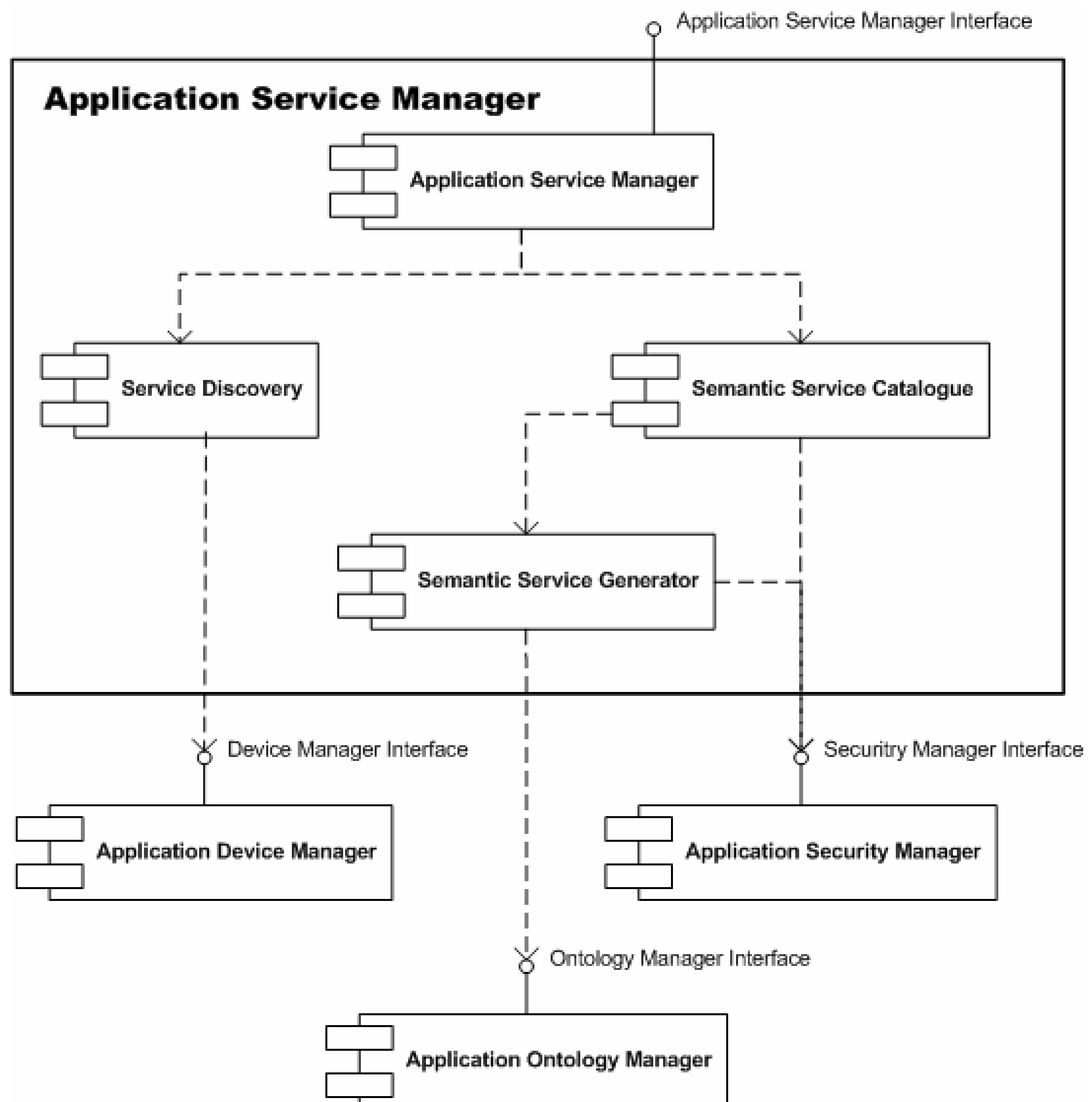

Figure 30 Application Service Manager

Purpose: The purpose of the Application Service Manager is to discover, create and execute semantic (web) services on top of devices. It adds a semantic layer above the Application Device Manager. Services might map to several device functionalities.

Main Functions:

- Service discovery
- Semantic service creation (service orchestration/clustering and mapping to device service)

Components:

Service Discovery Module: One of the major functions of the Service Manager is to discover new services in the network. This is taken care of by the Service Discovery Module. It will use the Device Manager to find out about services offered by different devices.

Semantic Service Catalogue: The Semantic Service Catalogue keeps track of and manages all service offered within one application. It can be queried about existing

services. It can also provide semantic service interfaces for the different services upon request.

Semantic Service Generator: The Semantic Service Generator is responsible for generating a semantic service interface for services offered by devices. It will create a software wrapper around the device services which other modules can use. The generated software will support a semantic-based service interface. It will support several semantic web standards, at least OWL-S and WSMO.

Dependencies: Application Service Manager, Application Ontology Manager and Application Security Manager

The following requirements are associated with the Application Service manager:

ID	Description
91	Any HYDRA device should have an associated description
92	Rule-based configuration of devices
104	Automatic Discovery of Services
111	Dynamic Web Service Binding
113	Composition (of services and devices)
114	Semantic enabling of device web services
119	Domain modelling support
120	Multiple Device Virtualizations
129	Support for Semantic Web Standards for Device Communication
157	Availability of combined services
158	There should be a hook-up-service
159	Service brokers must be organized in a hierarchical way
160	Search masks for device/service discovery
164	Support for Service standards
166	Trust based orchestration
196	Basic Service Registry
198	A service broker is responsible to provide services according to specific keywords
206	Middleware supports service discovery
207	Service selection by context
216	The middleware should have a graceful degradation service
290	Share service orchestration between users
291	Quality of Service as search criteria for service selection
294	Central service registry

325	Support aggregation and separation of devices and services
372	Interfacing with external systems
376	Security requirements must be part of the HYDRA MDA
392	Rules for selection of alternative devices

6.2.2.12 Application Storage Manager

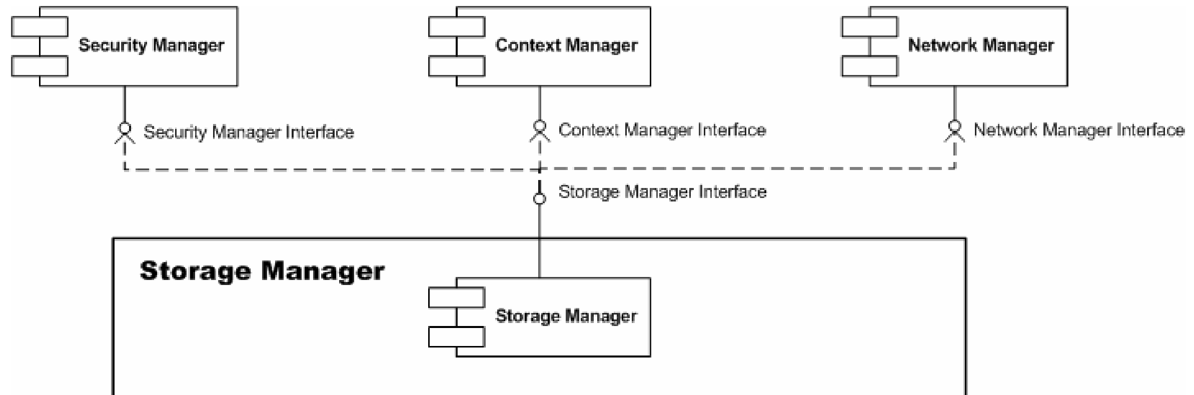

Figure 31 Application Service Manager

Purpose: The Storage Manager is responsible for storage management on the device level. It is needed to provide a single, consistent interface to all storage on the device. Depending on the device capabilities, the Storage Manager may handle the actual storage locally or take care of handling it through remote devices. Typical storage requirements include file and memory storage. Limited data sets to be used frequently and possibly by several cooperating devices, can be stored in a virtual shared memory structure to allow efficient and easy application programming, whereas big or less frequently used data sets are more suitable for file storage. In both cases it is the responsibility of the Storage Manager to provide a familiar interface to the storage and take care of the actual underlying file and memory operations.

Main Functions:

- Unique entry point for memory and file storage
- Control access to storage
- Handle mapping of virtual memory and files to actual local or remote storage
- Support synchronized (locked) access to storage

Dependencies: Network Manager (for remote Storage Manager access), Security Manager and Context Manager

The following requirements are associated with the Application Storage manager:

ID	Description
406	Storage Manager - Gateways information gathered storage
407	Storage Manager - Gateways information stored synchronization
409	Storage Manager - Device information metadata
443	Storage Manager - Gateways must allow efficient access to store data from associated devices

7. Development View

7.1 Overview

For the development viewpoint we explain the module organization in detail which shows the dependencies between modules. We also highlight how we handle revision control of source code files in a centralized repository and aspects of the development process to enhance the quality of the software.

7.2 Module Organization

The current iteration of the module structure model consists of two layers which contain the modules of the middleware on the one hand and the technological building blocks that are independent of the middleware on the other hand. Although the concept of the middleware modules is implementation agnostic we have to make the major design decisions before the implementation starts.

As described in chapter 6.2 the middleware is divided into device and application elements. Inside these two large building blocks we will implement certain modules in Java and others in .NET. The Device and Service Managers will be implemented in .NET and all other managers will be implemented in Java. A finer grained distribution of manager to devices will be described in chapter 8 in the deployment viewpoint.

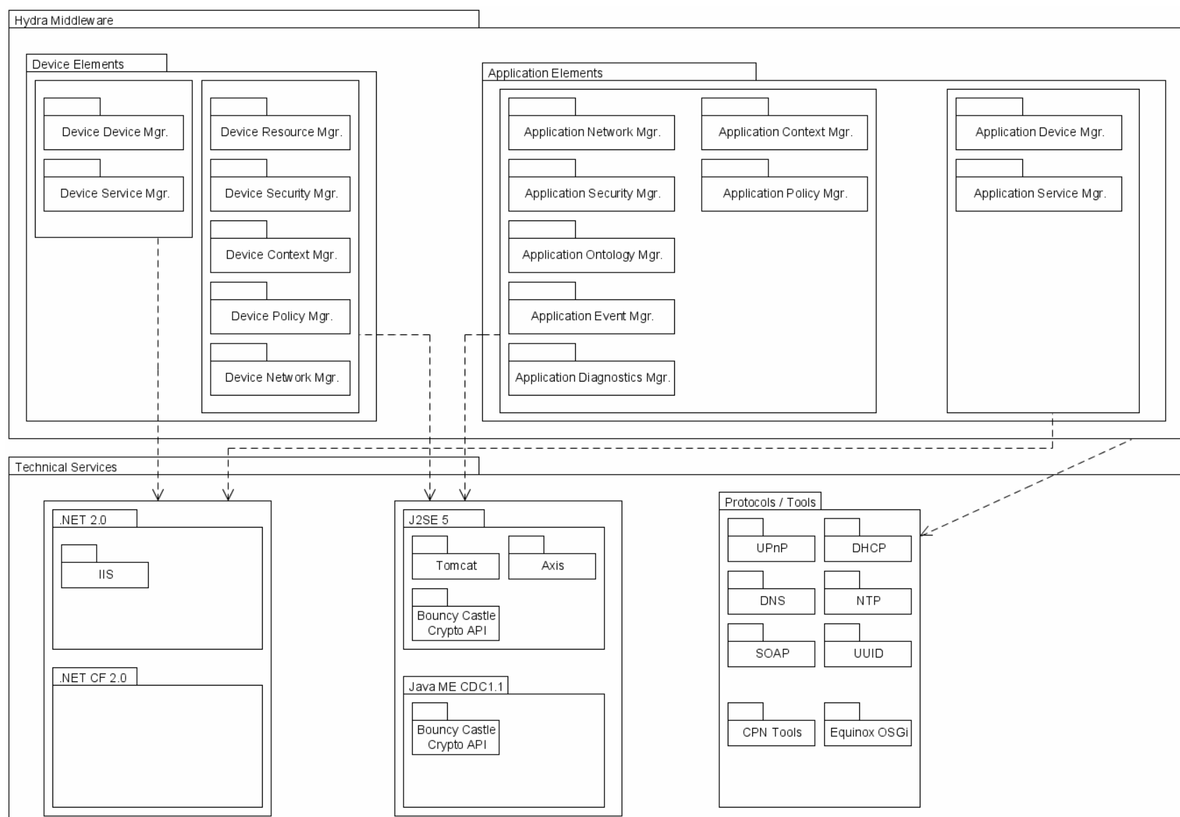

Figure 32 Module Structure

The middleware uses modules from a layer called Technical Services which define certain libraries and tools that will be used in the implementation phase. The modules are split up into Java and .NET components. The Technical Services layer also contains protocols and technologies that are used but need no implementation work from our side. Other tools are used as black boxes, which means that we only use the functionality without knowing the inner workings.

7.3 Configuration Management

An important part of modern software development is the Software Configuration Management. Configuration Management is often considered to have originated in the frame of software development in recent years and that it merely consists of a tool for versioning of source files. In software development the core of configuration management is made up of a version control system. Source code (or all source documents of the final product, respectively) is considered a system that spans time and space. Files and directories (which, of course, can contain files) form the space, while their evolution during development forms time. A version control system serves the purpose of moving through this space [Spinellis, 2003].

Note, that such time and space is not continuous, but rather discrete: Every revision corresponds to a snapshot of a document, which represents a certain state in its history of development [Westfechtel, 1996]. Version control systems are numerous with Bell Labs' SCCS (Source Code Control System) from 1972 being the first one [Bolinger and Branson, 1995]. Since then a multitude of other systems have been designed and used: CSSC, Revision Control System (RCS), Concurrent Versions System (CVS), Revision Control Engine (RCE), Subversion, BitKeeper, ClearCase, Forte Code Management Software, Serena Professional (originally PVCS), Perforce and Visual SourceSafe. [Thomas and Hunt, 2004; Reuter et al., 1996]

Configuration management contains version control, but in addition to this provides further methods of project management [Weischedel and Versteegen, 2002]. Locating configuration management between formal and complex procedures with rigid mechanisms, and collections of ad-hoc script based on simple version control is a subjective matter [Leblang and Levine, 1995]. Software configuration consists of software configuration items (SCI) which can be organized in three categories: computer programs, internal and external documentation to these, and data [Pressman, 2001].

A *baseline* is a version of a SCI on which developers and management or customers have agreed upon. A modification of this requires a formal change request. A *release* is a version which is provided to users, whereas a promotion is to be provided to fellow developers. Several problems for software configuration management can be derived from this [Ommering, 2003; Bruegge and Dutoit, 2000]:

Version control, i.e. filing of different versions of the same SCI

Temporal variation, i.e. allowing modification to occur in parallel, which can be discriminated into two types:

1. Small and bounded activities like fixing of bugs or implementation of a new feature and
2. more complex activities like creating a new program branch, which is developed largely independently of the main development branch (trunk).

Permanent variation, i.e. ability to create different products out of the same sources. These products are also called variants.

Build support, i.e. the build process (that is, building of the final software product from its sources) should be supported by configuration management. This is often accomplished through dedicated build tools like Make, Dmake, Imake, Odin, Ant or NAnt, although integrated tools like DSEE, ClearCase or Jason exist, too.

Distributed Development, i.e. access to the databases of the configuration management should be possible from all over the world. Necessary coordination tasks must also be supported.

The Institute of Electrical and Electronics Engineers (IEEE) sees configuration management as a discipline that uses observation and control on a technical as well as on an administrative level. Aims to achieve are:

1. identification and documentation of functional and physical characteristics of a SCI,
2. control of changes to these characteristics,

3. recording and reporting of change processes and implementation states,
4. and auditing of specific requirements.

This results in the following activities that constitute configuration management [Bruegge and Dutoit, 2000]:

1. identification of SCIs and their versions through unique identifiers
2. control of changes through developers, management or a control instance,
3. accounting of the state of individual components,
4. auditing of selected versions (which are scheduled for a release) by a quality control team.

Software configuration management deals with the governing of complex software systems [Westfechtel and Conradi, 2003].

For source code management we have set up a Subversion repository where all artifacts of the development process will be stored and organized. Subversion, like many other version control systems, manages different versions of documents by calculating editing operations between these and then only saving these operations. This usually leads to very good compression results.

We use subversion because it is a modern revision control system. It has been designed to resolve many of the short-comings of widely-used CVS, while still being free software. Subversion attempts to fix most of the noticeable flaws of the Concurrent Versions System (CVS) and is very powerful, usable and flexible. New to Subversion is the way how it internally treats directories and documents: directories (which are also documents) only contain links to specific revisions of other documents. Thus, when a source file is copied, Subversion will not copy all the data, but rather create a new directory entry referencing the same object. Hence, it is possible to copy the entire development branch at very low cost.

Access to the Subversion code repository is provided through secure SSL connections using our web server. Thus, this repository allows the geographically dispersed group of consortium members to collaborate and share their source code. It tracks changes to source code and allows a versioning of source code files. Additionally, Subversion remembers every change ever made to the files and directories so that earlier versions can be recovered in case that something was accidentally deleted.

The Subversion repository created for the Hydra project comprises three subdirectories:

- *branches*, i.e. for short time branches
- *tags*, i.e. a full copy of the source code for releases
- *trunk*, i.e. for the working source code as the current development version

For supporting the build process we agreed on using Ant for Java software components and NAnt for .NET components.

7.4 Development Decisions

The specific developments in Hydra are to be carried out by a group of partners working collaboratively most of the times, so one of the preliminary steps in the development process of Hydra is to harmonise the development methodology and the tools to be used in those developments. This is completely necessary in case we want to work in parallel and efficiently.

The project aims at the use of open source solutions where possible, but this is only a desirable requirement for choosing the appropriate tool and not a mandatory one. It is up to the developers of the consortium partners in their own organization to select the most appropriate development environment in Hydra depending on their own preferences and experience. We will not decide about any IDE to be used in particular.

The consortium took the decision to make use of the programming languages Java and C# for the implementation of the web services that provide the functionality of the different managers in the

middleware. Besides this restriction, the choice regarding the implementation language for different Hydra managers remains the responsibility of each WP, according to its preference or know-how. The two languages, among other features, provide many facilities for designing Web Services applications, apart from being well-known implementation languages. In detail, the consortium has decided that the managers will be using the J2SE 5.0 (Java 2 Standard Edition) and the J2ME (Java 2 Micro Edition) frameworks for Java developments. Moreover, and concerning C#, the consortium will make use of the .NET 2.0 and .NET CF 2.0 (Compact Framework) frameworks.

Although no specific IDE has been selected. Eclipse (for Java implementation), whose Web Tool Platform could favour web services development, and Microsoft's Visual Studio 2005 (for C# implementation) represent good and proven alternatives for software development.

Moreover, the Hydra consortium will use Ant, Nant and Maven as the automatic software building tools in the project. CruiseControl (see http://cruisecontrol.sourceforge.net/) and the unit testing suites JUnit and NUnit will be used for testing purposes. Finally, complementary to the Subversion system, JIRA is the tool chosen by the consortium to control and track bugs in the developed software.

Concerning Web Services development, we agreed to follow the WS-I Basic Profile (BP) specifications. The WS-I BP is a specification from the Web Services Interoperability industry consortium (WS-I), which provides interoperability guidance for core Web Services specifications such as SOAP, WSDL, and UDDI. The profile uses the Web Services Description Language (WSDL) to enable the description of services as sets of endpoints operating on messages.

7.5 Test Plan

Quality assurance specifically deals with the management of quality. In this term "assurance" means that under the condition that the processes are followed, the quality management can be assured regarding product quality. Furthermore, quality assurance strives to encourage quality attitudes and discipline among the developers and the management. Quality assurance encompasses software configuration management, quality control and software testing.

Software testing can be used to verify the implementation of functional requirements and hence is a popular strategy of risk management. To some extent software testing can also be used to find errors, however, testing cannot prove the absence of errors in software as testing is only as good as its test cases. Furthermore, by the time that testing occurs, it is too late to build quality into the product [Lewis, 2005].

According to the IEEE the test plan is part of the test documentation. It prescribes the scope, approach, resources and schedule of testing activities. Thus it specifies which items and features are to be tested and how, who is responsible for testing and what associated risks are [IEEE Std 829-1998]. A concrete test plan was already specified in D8.1. In this section we want to stress that the actual testing should particularly focus on things that are important to the different stakeholders of the software. Hence the initial scenarios already provide highly useful information on the preferable focus of testing activities [Rozanski and Woods, 2005].

Besides the test plan, the test documentation contains specifications of actual tests and reports of their application [IEEE Std 829-1998]. In HYDRA we plan to have the test specifications evolve along with the software. The same holds for the tests of unit testing, which constitute a formalization of test specifications that can be executed by machines. Test reporting will be kept as non-bureaucratic and transparent as possible, rather focusing on immediate feedback so that bugs can be tracked down faster and more easily. For example, automatically generated reports of failed integration tests instantly inform developers via email.

Finally, we want to stress that for creating high quality software it is not enough to follow a high quality process (including testing), but also to utilize modern and sophisticated technologies, and to employ motivated, capable and well-trained personnel [Curtis et. al, 1995]. We commit ourselves to reacting appropriately to such challenges.

7.5.1 Continuous Integration

In order to ensure that changes inside software modules do not affect related modules we will create unit tests which can be executed in order to check that certain functionality is performed correctly. These test cases can be run in an automated fashion so that the developer can easily validate the functional integrity before committing his changes to the central Subversion repository.

Certain agile methodologies like Extreme Programming (XP) propose writing test cases even before implementing the actual code. We leave this up to the developer because in a research project we are sometimes forced to implement functionality that is not well understood. In this case writing unit tests before the code does not make sense. In prototyping situations the developer needs more freedom in trying out several options before the final design decision. In these situations the developer has to add test cases after he made the decision.

Continuous Integration is also a term that has its origin in the concepts of XP (see http://martinfowler.com/articles/continuousIntegration.html). In this community it is common practice to immediately commit every change to the revision control system, no matter how small it is. The rationale behind this is that other developers should always work with the latest version of the code base. Furthermore, Continuous Integration describes a process of executing all test cases in a project in an automated and frequently repeated way. Failures in test cases will be automatically detected and reported to the persons that committed the changed source code files since the last successful test run.

What Continuous Integration is and how it works can be shown by giving a quick example of the process of implementing a small feature, which requires only some few hours of work.

Continuous Integration increases the awareness of problems by involved developers by reporting back problems in a very short time frame after these problems occurred. The error resolving takes less time and efforts compared to failure detection in later stages of the development. The first step is to take a copy of the current integrated source onto the local development machine. This is done by checking out a working copy from the mainline (or trunk) from the revision control system. This local copy can now be modified so that it implements the required feature. In the Continuous Integration paradigm, implementing does not only mean to make changes to the actual code, but it is also required to add unit tests for automated testing of the feature. This is sometimes called self-testing code (see http://www.martinfowler.com/bliki/SelfTestingCode.html). Note, that changing the local copy implies the later modification of production code.

The developer, who changes the system, makes every effort to ensure that her feature does not introduce a bug: Once she is done (and usually at various points while she is working) she carries out an automated build on her development machine. This takes the source code she is working with, compiles and links it into an executable, and runs automated tests. After that the developer performs manual tests. Only if everything builds and tests without errors is the overall build considered to be good.

With a good build, the developer can then think about committing her changes into the repository. The twist, of course, is that other people may, and usually have made changes to the mainline in the meantime. So first the developer updates her working copy with the changes from the other developers. If the other developers' changes clash with hers, it will (most probably) manifest as a failure either in the compilation or in the tests. In such a case it is the developer's responsibility to fix this and repeat the above steps until her working copy is a good build that is properly synchronized with the mainline.

Once this state is reached, the developer can finally commit her changes into the mainline by committing her changes into the repository. However this commit does not finish her work. At this point, another build is required; but this time on an integration machine based on the mainline code. Only when this build succeeds it is said that the changes are done. The reason for this is that there is always a chance of missing something on the local machine or that the repository was not properly updated. Only when the committed changes build successfully on the integration machine the job is done. This integration build could be executed manually by a person, or be done automatically by a Continuous Integration tool.

If a clash occurs between two developers, it is usually caught when the second developer builds her updated working copy. If not the integration build should fail. Either way the error is detected rapidly. At this point the most important task is to fix it, and get the build working properly again. In a Continuous Integration environment you should never have a failed integration build stay failed for long. A good team should have many correct builds a day. Bad builds do occur from time to time, but should be quickly fixed.

The result of doing this is that there is a stable piece of software that works properly and contains few bugs. Everybody develops off that shared stable base and never gets so far away from that base that it takes very long to integrate back with it. Less time is spent trying to find bugs because they show up quickly. The advantages of Continuous Integration can be summarized as follows:

- When unit tests fail, or a bug is discovered, developers might revert the code base back to a bug-free state, without wasting time debugging.
- Integration problems are detected and fixed continuously - no last minute hiatus before release dates;
- Early warning of broken or incompatible code;
- Early warning of conflicting changes;
- Immediate unit testing of all changes;
- Constant availability of a "current" build for testing, demo, or release purposes;
- The immediate impact of checking in incomplete or broken code acts as an incentive to developers to learn to work more incrementally with shorter feedback cycles.

All kinds of scheduled builds, if done frequently enough, would fall under Continuous Integration. However, nightly builds are too infrequent to qualify. Notable examples of Continuous Integration software include:

- CruiseControl (see http://cruisecontrol.sourceforge.net/)
- Borland Gauntlet (see http://www.borland.com/us/products/silk/gauntlet/index.html),
- AnthillPro (see http://www.anthillpro.com/html/products/anthillpro/),
- TeamCity (see http://www.jetbrains.com/teamcity/),
- Parabuild (see http://www.viewtier.com/products/parabuild/feature_overview.htm)
- Pulse (see http://www.zutubi.com/products/pulse/).

7.5.2 Code Inspection

"Now there's no guarantee that you get all the integration bugs. The technique relies on testing, and as we all know testing does not prove the absence of errors. The key point is that Continuous Integration catches enough bugs to be worth the cost." [Fowler and Foemmel]

The achievements of open source software show that is possible to collaboratively create software of high quality without a highly paid management. In this case the programmers are motivated intrinsically; there is little extrinsic reward for their efforts [Mackus et al., 2002]. Their rewards are commendation, feelings of having done something for the community being proud of "their" software.

In commercial software development, there is extrinsic reward (primarily in the form of payments made by the employer), intrinsic motivation is not always given. Therefore creating high quality software in itself might not be the primary goal of developers, because they get paid for reaching milestones. Quality, however, is almost inescapably not a milestone, because it depends (at least partially) on the people getting in contact with it: developers as well as maintainers and users. It is cardinally a soft attribute based on the perception of humans. As a result, it is basically impossible to measure quality as a whole automatically [Kelly and Shepard, 2002].

It would be best to create intrinsic motivation in developers, but how to achieve this – even if management is committed – is an open question. Traditionally this is countered by establishing rigid control mechanisms, for example the ISO-9000 standards or coding rules. In our opinion ISO-9000 is far too rigid for a research project like HYDRA. But a set of coding rules, which forbid or promote specific behavior, has been developed and published as a live document in the project Wiki. We are steadily improving these to prevent bad code quality.

But even sensible coding standards tend to be overseen or ignored when facing time pressure [Sutter and Alexandrescu, 2005]. Moreover there is no direct feedback from the rules themselves, what would be a precondition for learning. Because of that, our coding standards will be accompanied by software metrics, which automatically verify adherence to coding rules. For this we will rely on the open source CheckStyle library, whose tests range from trivial file size or formatting checks to hidden field detection and magic number tests. However, metrics work on the lowest level of code far from the semantic level. As pointed out before, measuring quality is a hard task, because all-embracing hard criteria for measuring (code) quality automatically cannot be set due to the soft nature of quality.

We are currently evaluating possibilities for installing a code reviewing process for improving code quality of HYDRA sources. However, such a process would have to be different from traditional reviewing so that it better suits the agile development approach of HYDRA. What we want to achieve is that code quality is constantly monitored; both, by software metrics and by fellow developers. Through a sportive quality competition, individual developers will be guided towards a common coding style and they will develop a feeling of quality code.

Our approach for the Hydra project is to provide them with the ability to assess any piece of code at any time and to anonymously give the authors feedback on what they feel of the quality of the code. Accountability for parts of the code and timelinesses of reviews will be determined automatically and allow every individual developer to capitalize on the team's perception of the individual's code.

7.5.3 Relation to the Test Plan defined in D8.1

This chapter extends the views of D8.1 by giving detailed background information on the various strategies that are applied in software quality management. Decisions concerning the tools to be used, which were made in D8.1, are justified and viewed from a broader context of software engineering.

This chapter fills the gaps in D8.1 that exist between the background information and the tools it proposes. The earlier deliverable depicted quality as a set of criteria and seems that testing is the one and only factor on the way to high quality. This chapter rectifies this view and emphasizes the need of addressing quality through several established techniques. It additionally includes configuration management, code inspection, coding rules, a people view and our commitment to flexibly react to forthcoming challenges of quality assurance.

ISO 9000 often plays an important role in industrial production settings. It was therefore necessary to state what our position towards such standards is and note that we deliberately chose agile and not so heavyweight approaches.

8. Lessons Learnt from the First Iteration

8.1 Overview

This chapter documents the outcome and experiences gained from the realisation of this first prototype in the form of "Lessons Learnt". The lessons learnt reflect the first phase of the Hydra project. Throughout the course of this phase new questions and ideas arose, which might trigger a continuation of the project with specialized foci on these new issues. For the next iteration of the developments, these lessons learnt will have strong impact on the next phase of the requirement engineering process. Regarding the software architecture documented by this deliverable, the partitioning and the number of the different managers can be investigated as well as their functionality can be evaluated against the requirements.

8.2 Concept Prototype

The conceptual work accomplished in Hydra so far has been proven by a first prototype of the Hydra middleware. It was based on use cases and scenarios taken from the building automation domain (see D9.5 for more details). This prototype consists of components implemented by the members of WP4 to WP7 and the integration of these components into a first prototypical common Hydra middleware has been done by members of WP8. Based on this middleware the members of WP9 designed and developed the user application for building automation.

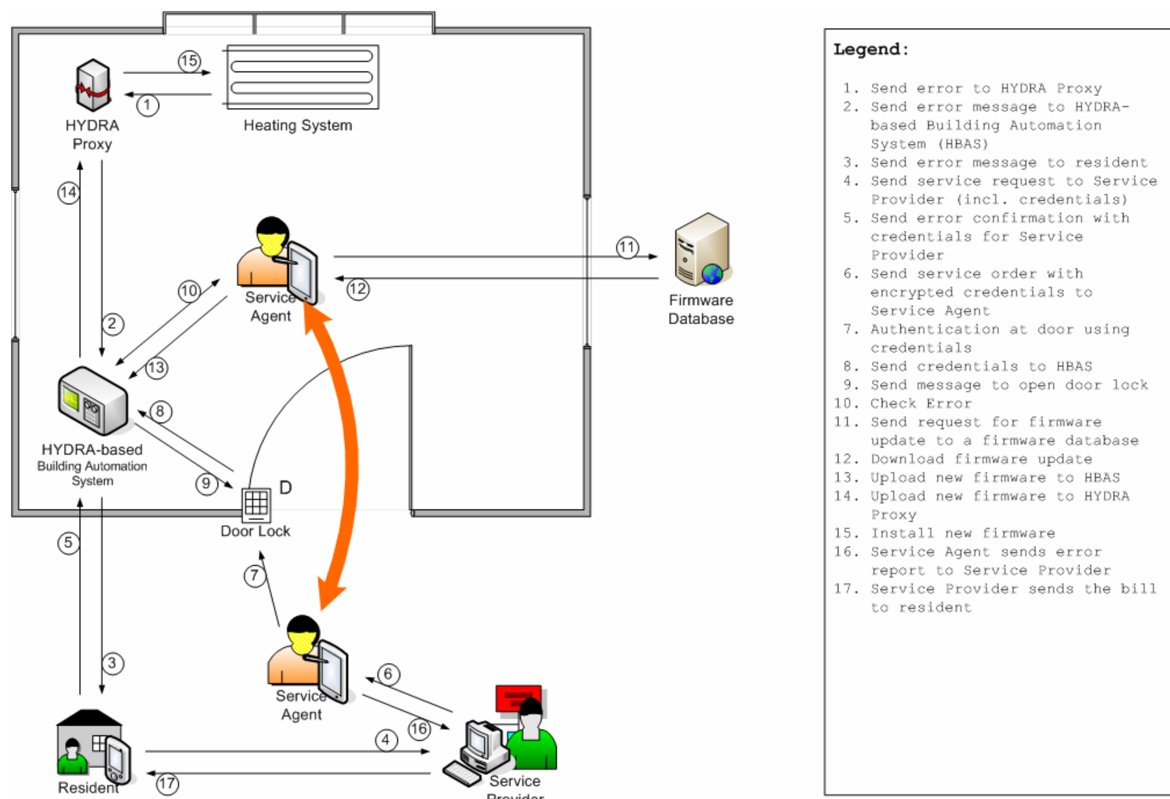

Figure 33 Sequence of steps for technical scenario

The scenario that has been demonstrated at the review meeting was taken from the Building Automation domain (cf. deliverable D2.1). Figure 30 illustrates the steps, into which this scenario has been broken down, and gives an overview of the interaction of the involved persons and systems. The precise distribution of software modules to hardware devices and the necessary network infrastructure has been described in more detail in the deployment view chapter of

deliverable D3.3. This concept prototype conveyed a better understanding of the system and helped us discover potential building blocks of the system. Therefore, this prototype serves as a valuable source of experiences and lessons learnt during its development. The following section reports on such lessons learnt and clusters them into six major categories.

8.3 Categorization of Lessons learned

Issues referring to experiences gained during the first iteration of the Hydra project have been systematically collected from the partners in the consortium. Subsequently, these issues have been assessed and condensed into a list of lessons learnt addressing different aspects of the developments. However, a clear separation of these aspects has not been possible, which manifests itself in some overlapping between the addressed aspects.

8.3.1 Communication and Networking

During the preparation phase of the demonstrators, the integration of the different software modules revealed that different methods for the communication between the managers had been used. For the handling of the communication between the managers, some solutions preferred the use of events, whereas other solutions base on direct calls of the manager. The communication and flow of information between the managers still remains an open issue. The project consortium is currently investigating different approaches towards orchestration.

During the development the problem came apparent that the uniform resource locator of the respective managers really needs to be uniform even if different network technologies and communication methods are used. Some mobile phones such as the Nokia phone cannot be on a local network while they are using GPRS as a means of communication. This problem implies that public IP-addresses need to be considered.

Regarding the discovery of devices and services the integration phase yield to the result that the UPnP device discovery works well and that it consumes maintainable resources on devices. However, no experiences have been gained regarding the scalability of this discovery mechanism, since the demonstration did not involve the discovery of large amounts of devices. Furthermore, it became clear that a type of session management will be needed, which could for instance simplify security related solutions. Therefore, the role of the Network Manager is currently reconsidered and the session manager has been integrated into the Network Manager.

In summary, the lessons learnt regarding the communication and networking basically raise new issues concerning:

- Unification of communication methods
- Standardization of manager identification
- Reconsideration of the Network Manager role

8.3.2 Event Management

In conjunction with the lessons learnt regarding the communication and networking, some problems during software integration were attributed to the event management, which has been overly used in the current prototype. The application of JAX-WS and Axis for event-driven application worked fine, although some latency has been identified due to multiple concurrent function calls. In addition, the use of web applications as Event Manager in the role of both publisher and consumer works fine. However, the development of web applications for small devices such as PDAs limits the usage of HTML, JavaScript and CSS. Further challenging issues such as the expiration of subscriptions to events uncovered the need for debugging facilities in order to trace the flow of events.

In summary, the lessons learnt regarding the event management reveal a positive experience regarding the technical aspects, although the following issues require further investigation:

- Specification of the information flow between the managers

8.3.3 Interoperability

The preparation of the prototype allowed for gaining experiences regarding the use of multiple programming languages and enabled a testing of the interoperability of web-services provided by several implementations. Altogether, the interoperability of the web-services between different platforms and programming languages worked fine. .NET and Java-based web-services could co-exist in the prototype. However, some problems occurred with namespaces and Limbo because Limbo did not handle namespaces correctly. This problem should be fixed in the next iteration.

The multilingual implementation of the prototype revealed that the software architecture should be fairly independent of any specific programming model and it should be possible to implement managers in either programming model. However, this issue need to be observed and monitored in the iterations planed for the future. In this regard the question has been raised whether it will be possible to use higher level languages in order to be independent of any programming language, so that the user can select only one language for programming applications

In summary, the lessons learnt regarding the interoperability of web-services and programming languages were throughout positive. However, some issues involved with the use of web-services regarding the device-internal communication are reported in the following section. The further evolvement of the interoperability needs to be monitored in the future iterations.

8.3.4 The Use of Web Services

As a positive result from the prototype the use of web services on embedded devices can be reported. The prototype demonstrated that there is sufficient resource usage of Limbo-generated service on Java ME and Java SE devices (details on resource usage profiles will be reported in deliverable D4.2). However, deeper analysis and research need to be undertaken on which further devices web services can be used, e.g. the Nokia N80 with current implementation requires the usage of the Nokia Raccoon gateway which makes interaction with the mobile phone slow.

Although the approach to implement the service-oriented architecture by using state of the art Web Services technology gives the required freedom to design a common, consistent and convincing middleware architecture for future ambient environments, a deeper and more concrete analysis of the efficiency and performance need to be performed. The architecture currently comprises a high number of managers, which even multiplies due to their instantiation for a specific application. Therefore, the average invocation time of a web-service and the overall latency need to be calculated carefully in order to deliver acceptable results and meet the performance requirements. Furthermore, the demand for an analysis of the feasibility of web services related to the installation and deployment effort became apparent. An internal report should document and quantify the results and experiences gained for far, and provide required additional results.

In summary, the use of web-services showed positive results. However, the following issues need to be investigated in future iterations:

- Investigation on the performance and efficiency of web-services
- Research on the need of distinguishing direct calls and web-services
- Revise the purpose and functioning of all managers

The web-services architecture is a well established 'standard' and its feasibility or infeasibility for the Hydra middleware should be proven carefully and convincingly.

8.3.5 Use of Ontologies

So far, less hands-on experience with the use of ontologies could be gained during the first phase of the project. In terms of creation and administration of the Hydra ontologies, the means for ontology modelling and reasoning, in particular which language (e.g., SWRL) and engine (e.g., Pellet), are currently investigated. The implementation of the SWRL (Semantic Web Rule Language) as a proposal for a Semantic Web rules-language, combining the Web Ontology Language (OWL DL and

Lite) with the Rule Markup Language, seems to be incomplete. This would raise problems regarding its use for general ontological reasoning and for the configuration of Limbo.

Regarding the operational usage of ontologies there is a discussion in progress whether or not a basic ontology should be provided, which links with application-specific ontologies, e.g. though using the "import" statement. In addition, the advantages and disadvantages of one centralized ontology manager are deliberated.

While discussing the areas of potential application of ontologies and ontology reasoning capability, the demand for domain experts arises who possess comprehensive expertise for the creation of reasonable ontologies like for instance for the devices. Furthermore, the awareness rose that the set of Hydra ontologies can be designed with reasonable efforts once the user requirements further substantiate. Thus, once the possible contents of the required models have been determined, issues like the ones mentioned above can be approached.

In summary, the lessons learnt regarding the use of ontologies basically address the following issues:

- More experience with the application of ontologies required
- Investigation of methods and tools for the administration of ontologies in progress

8.3.6 Separation of Application and Middleware

After finishing the integration work for the prototype, a strong meshing of middleware- and application-/scenario-specific source code has been observed. The narrow operating schedule basically caused the missing clear separation of application and middleware. In particular, the predefined interaction between the managers turned out to be scenario-specific. In the current implementation the orchestration of the managers is determined by the managers themselves, however, the orchestration of managers need to be determined by the application. Therefore, a new Orchestration Manager has been introduced into the architecture (cf. Section 6.2.2.7) in order to handle task.

The same holds for the interoperation of the Hydra middleware and external existing applications such as third party software modules. The upcoming decision on whether or not to apply an open source license model affects the usage of third-party libraries in the Hydra middleware.

In summary, the lessons learnt regarding the separation of application and middleware:

- Elaboration of a clear separation of Hydra middleware and application

8.4 Implications on the Software Architecture

The next iteration of the software architecture needs to critically analyse and explicitly address the list aspects summarised in the previous section. The current iteration already started with a revision of accordant requirements and the specification of new requirements. This section lists a set of key implications of the lessons learnt on the software architecture and on the work to be accomplished in WP3. The addressing and implementation of these implications will be reported in the deliverable D3.9 "Updated System Architecture Specification", which is planned for month 22.

8.4.1 Introduction of an Information View

The lessons learnt specifically uncovered a problem regarding the communication of the managers and the information exchange between them. The next iteration of the software architecture needs to provide a complete but high-level view of the static information structure and the dynamic information flow between the managers. Thus, the introduction of an information view will at an architectural level answer questions about how the middleware will store, manipulate, manage and distribute information. This specification will address the need to establish a common way of performing manager communication.

The introduction of an information view on the software architecture requires the modelling of data in order to illustrate and further specify the composition of the middleware managers and the communication between them. The types of models relevant for the Hydra middleware architecture comprise:

- *Static Data Structure Models* describing what kind of data the managers need for internal use and how this data looks like, i.e. the important data elements and the relationships among them.
- *Information Flow Models* describing which data is exchanged between the managers and their internal components, i.e. the dynamic movement of information between elements of the system and with the outside world.
- *Data Ownership Models* describing which component is responsible for which data.
- *Data Lifecycle Models* describing the transitions that data elements undergo in response to external events, i.e. the way data values change over time.

As a first step of the realisation of the information view on the software architecture, the component diagrams described in Chapter 6 needs update and refinement. Therefore, it is necessary to rethink the number of managers of the current version of the architecture. The managers should be consolidated, i.e. rearranged or merged, in order to reduce the amount of managers rather than devising additional ones. Each manager needs to be described in much more detail concerning the components they comprise and their interconnection to other managers. The definition of interfaces is required as well as a clearer description of the responsibilities of each component.

In conjunction with the refinement of the manager descriptions, an agreement on the means applied for the description of the interaction between the managers will take place, e.g. Sequence Diagrams, Collaboration Diagrams, Activity Diagrams, etc. This procedure will involve a discussion on how to handle incompatible data and on the scalability, i.e. how much data will be generated and used.

8.4.2 Introduction of a Common Deployment View

The revision of the managers comprising the software architecture and their internal components as well as the introduction of an information view on this software architecture entails a revision and generalisation of the deployment view. The preceding deliverable (D3.3) on the software architecture introduced a deployment view specifically addressing the realisation of the prototype as a first demonstrator of the Hydra project taken from the domain of building automation. The next iteration needs to produce a deployment view that describes on a common level the environment into which the system will be deployed, including the dependencies the system has on its runtime environment.

The deployment view focuses on aspects of the system that are important after the system has been tested and is ready to go into live operation. This view defines the physical environment in which the system is intended to run, including:

- Required hardware environment (e.g. processing nodes, network interconnections, etc.)
- Technical environment requirements for each node
- Mapping of software elements to the runtime environment
- Third-party software requirements
- Network requirements

The deployment view needs to document the required deployment environment of the Hydra middleware that is not immediately obvious to all of the interested stakeholders. This regards the selection of managers required the operation of the desired application and the choice of the platform the respective manager will run on. Three models will constitute the central description of the deployment view:

- Runtime platform model

- Network model
- Technology dependency model

The runtime platform model defines the requirements on the computing nodes, which nodes need to be connected to which other nodes via network (or other) interfaces and which software components run on which node. Typically, the runtime platform model is captured as a network node diagram that shows different kinds of nodes, the interconnections required between them and the allocation of the software elements between the nodes.

The network model complements the runtime platform model regarding the details of the network. In Hydra the underlying network is complex and therefore, it needs to be described in a separate (but related) network model. The purpose of the network model is to define what types of network connections will be supported and if there are constraints that have to be adhered to during implementation and network design.

The technology dependency model captures the dependencies of the Hydra middleware components on other software they rely on regarding licensing, costs, efficiency, etc. In further iterations of the implementation the partners of the Hydra consortium will acquire a more and more complete picture of the technology their software depends on. Usually, technology dependencies are derived from the development view and described on a node-by-node basis in a simple tabular form.

8.4.3 Improved Service-Oriented Architecture Approach

The use of several different means of communication has been reported in the lessons learnt section. Similar issues concern the use of web-services, since web-services introduce additional latency into the system, which might not be acceptable in some cases. Therefore, based on a systematic analysis of the performance and efficiency of web-services, a strategy of a potential distinguishing of direct calls and web-service calls should be elaborated and the service-oriented architecture approach should be improved.

Even the extensive usage of services in a heterogeneous environment comprising different devices, platforms and programming languages not every function call is required to be a web-services call. A potential function call strategy could be that the managers or their internal components use direct calls such as library calls, RMI, .NET remote, etc., if they are located on the same platform, and that they use web-service calls, if they are on different platforms. However, this strategy would require an additional control mechanism that dynamically decides whether the managers are located on the same or different platforms. This issue still remains to be discussed and means of quantification and measurement will be needed for a final decision.

Regarding the composition and orchestration of web-services the services-oriented architecture needs to be improved as well. In this regard service composition means the aggregation of basic services in order to gain higher-level services. For the next iteration an investigation on whether or not the middleware will support service composition needs to be undertaken. In addition, requirements for service composition need to be identified. The resulting impact on the software architecture will be the addition of new components that fulfil the respective requirements and provide functionality to handle service composition and orchestration.

8.4.4 Integration of Context-Awareness

Currently, activities of the consortium concern the realisation of context-awareness in the Hydra middleware. Besides the elaboration of a conceptual framework comprising for instance a definition of context, a Context Manager has been specified in the first iteration of the project. For the second iteration this functionality needs to be an inherent part of the prototype. Since the Context Manager is considered to be an omni-present entity in Hydra, a general architectural approach for the realisation of context-awareness in the Hydra middleware is required.

Traditionally, the middleware-based realisation of context-awareness comprises comprehensive software libraries and employs methods of encapsulation for the separation of business logic and functionality of context-aware computing. A context middleware introduces an own layered

architecture of context-aware applications with the intention of hiding low-level functionality for the acquisition, transformation, dissemination of context information. In addition, current middleware-based approaches ease extensibility and simplify the reusability of software components.

However, these approaches provide weak support of programming abstractions like the determination of situations or the access to historical context information. Furthermore, a context middleware mostly focuses on the sensor-based acquisition of context information, and thus, disregard potential input of users and insufficiently address control and actuation mechanisms of the targeted context-aware application. Hence, middleware-based approaches omit covering the end-to-end process chain of context-aware application ranging from the acquisition of context information to the realization of adaptations at multiple levels of abstraction.

The current design of the Context Manager primarily focuses on spatio-temporal context information such as location, proximity or presence. This set of possible context information needs to be greatly extended and determined in order to meet the needs of application and device element. Based on determination an accordant model of context can be defined. The deliverable D3.8 "Context-Awareness Report" will document the results of the ongoing discussions.

In this regard a concept needs to be elaborated on whether a centralised, decentralised or hybrid approach of context modelling will be followed by the software architecture. In this connection, the communication structure required for the sensing and access of context needs to be aligned with security and privacy requirements. A distributed management of context where each entity, i.e. application, user, device, etc., keeps its own data has the advantage that a sharing of data can be initiated only if necessary. Thus, the user or her respective delegate device stays in control of the context information. However, a new approach of linking context information needs to be elaborated, e.g. for the generation of higher-level context information. In this regard, the decisions concerning the use of ontologies in the Hydra project need to be taken into consideration as well.

9. Guidelines and Constraints

The following chapter is very general and will be continuously refined in the next iterations. Here we will collect which guidelines and constraints have been identified. For the guidelines we will reference which ones we have created and they will be done in separate documents. The constraints will be reflected in the architecture and we will reference them accordingly if there is a clear relationship. Otherwise we will make sure that they are properly addressed in the architectural design.

9.1 Guidelines

In order to support the developer in the creation of constraint compliant applications we will provide guidelines on how to effectively and efficiently implement specific parts of an application. We plan on having a model-driven approach in a distributed loosely coupled architecture so we will need to create guidelines on how to model constraints for the various areas. For example the modelling of the security aspects of an application what aspects of the security aspects can be modified and how this will influence the runtime application will be very helpful. The guidelines should also contain checklists so the developer is capable of easily check if all necessary steps have been done.

We will have to discuss in further iterations in which areas we will create such guidelines and also test the usefulness of them.

9.2 Constraints

Application developers have to ensure, that their applications are compliant to regulatory constrains or other (industry) standards. The Hydra middleware is designed to be applicable in different application domains which intentionally include the usage in an international context. Constraints on Hydra applications therefore can be very complex and depend on many factors which are outside of the control of the developer. Data protection directives, for example, should be considered by developers of building automation applications and vary between countries.

9.2.1 General Constraints

This paragraph will be refined in later iterations. The following constraints have been identified to generally applicable to various domains:

Data Protection: In all Hydra application domains, the exchange of information plays a central role. Different user groups provide personal information that is processed by a Hydra system. To protect the users' privacy, the compliance with data protection directives is crucial to the use of Hydra applications.

Safety at work: Professionals, who use Hydra based systems in their working environment, must be protected according to the holding conditions for safety at work.

Environment protection: Electronic devices consume electrical power at runtime. But also the production process consumes energy and other resources. Broken devices must be disposed or recycled. Environment protection standards have to be considered when building Hydra systems.

Radio spectrum policy: Hydra supports a variety of wireless communication protocols. Not only the division of the radio spectrum but also the use of certain frequency bands is controlled by directives.

Copyright issues: Digital content, that is copyright protected, must be treated accordingly. Digital content is not only digital media like images, audio recordings or video, but also software products.

9.2.2 Domain Specific Constraints

This is a first draft of domain specific constraints for the three chosen domains we want to support in Hydra.

In the field of building automation, national-wide regulations for the safety and energy efficiency of buildings exist.

In health care applications, special quality guarantees are needed. International initiatives work out standards for medical devices and software development in health care.

The European agriculture industry is subjected to several regulations including consumer food safety, animal welfare and nutrition as well as the environment.

10. Conclusion

The software architecture presented in this deliverable is based on an iterative approach for the software architecture design process. In order to design an adequate architecture it is absolutely necessary that one gathers high-quality requirements from stakeholders. We have presented our analysis of stakeholders and a technical scenario for the Building automation domain in chapter 2. We have chosen to analyze only the Building Automation domain in this first iteration of the Hydra project.

In the following chapters we presented the overall methodology of our design process which is based on the process proposed by Rozanski and Woods (2005) and for the artefacts we produce during the process we have chosen to follow the recommendations from the IEEE standard 1471 as well as the notion of architectural perspectives again from Rozanski and Woods. The iterative process we have chosen includes the requirements elicitation with engaging the stakeholders and then designing/refining the architectural description. The input for the architecture are the requirements from the stakeholders that have been captured in WP2 and can be found in D2.5 Initial Requirements report and it must be noted that requirements and architectural design influence one another. After the revision of the architecture has been finished the process will continue with the implementation phase which is not part of this workpackage and out of scope for this deliverable. The different viewpoints and their relationships between them have been described and the term "architectural perspective" has been introduced. In chapter 5 we have highlighted several general design considerations.

In chapter 6 we illustrated the current definition of the software architecture by first presenting an overview by means of a diagram of the identified layers for the application and device elements. Continuing with that we presented the components in detail and described their purpose and how they relate to other components. We have assigned requirements to each component that completely or partially satisfy these requirements. Compared to the initial version of the architecture description the Device Quality of Service Manager has been added and the requirements associated with each manager have been updated.

Chapter 7 has been newly introduced in this version of the software architecture description. This chapter illustrated the development view on the architecture comprising the module organization and the dependencies between modules. It also described the means we apply in order to handle revision control of source code files. In addition, a strategy of ensuring the quality of the software along the development process has been determined.

Chapter 8 is also new in this version and reported on the experiences gained and the lessons learnt during the development phase of the building automation prototype. This chapter already initiated the revision of the requirements and their concrete implications will be documented in the next version of the software architecture.

In chapter 9 we present guidelines and constraints that we have identified. The contents of this chapter are very general and show the extension paths for the next iterations. The guidelines will be collected in this chapter and described but the complete guidelines will be separate documents. The guidelines should show how to use or configure the middleware, the SDK and the IDE to support the developer to work efficiently. The constraints are divided into common ones and specific constraints regarding the Building Automation domain and will be refined in the later iterations.

In the next iteration we will continue to expand the deliverable in terms of adding more viewpoints and refining the requirements with input from the other domains. The revision of the requirements will be reflected in the software architecture and result in a much more detailed description of the managers. In addition, the communication and the (scenario-unspecific) information flow between the managers will be documented.

11. List of Figures

Figure 1 Architecture Definition Activities 10
Figure 2 Architecture Definition Activities details 11
Figure 3 Viewpoint catalogue 12
Figure 4 Definition of Architectural Perspective 13
Figure 5 Architectural Perspectives and Views 13
Figure 6 Middleware Layer 16
Figure 7 Generic Middleware Stack 16
Figure 8 Software architecture layers 17
Figure 9 Overview of the Device Elements 18
Figure 10 Device Context Manager 19
Figure 11 Device Device Manager 20
Figure 12 Device Network Manager 22
Figure 13 Device Policy Manager 24
Figure 14 Device Quality of Service Manager 25
Figure 15 Device Resource Manager 27
Figure 16 Device Security Manager 28
Figure 17 Device Service Manager 30
Figure 18 Device Service Manager 31
Figure 19 Overview of Application Elements 32
Figure 20 Application Context Manager 33
Figure 21 Application Device Manager 34
Figure 22 Application Diagnostics Manager 37
Figure 23 Application Event Manager 39
Figure 24 Network Manager 41
Figure 25 Application Ontology Manager 43
Figure 26 Application Orchestration Manager 45
Figure 27 Application Policy Manager 46
Figure 28 Application Schedule Manager 48
Figure 29 Application Security Manager 49
Figure 30 Application Service Manager 51
Figure 31 Application Service Manager 53
Figure 32 Module Structure 54
Figure 33 Sequence of steps for technical scenario 61

12. References

Beringer, J. (2004): End-User Development: Reducing Expertise Tension. Communications of the ACM, 47 (9), pp. 39-40.

Bolinger D.; Branson T. (1995): Applying RCS and SCCS. O'Reilly, 1995, p.540.

Bruegge, B.; Dutoit, A.H. (2000): Object-oriented software engineering: Conquering complex and changing systems. Prentice Hall.

Curtis, B., Hefley W. E., Miller, S. (1995): Overview of the people capability maturity model. Technical report, Software Engineering Institute - Carnegie Mellon University.

Day, J. and Zimmerman, H. (1983): The OSI Reference Model. In Proceedings of the IEEE, (71)12:1334-1340, December

Dertouzos, M. (1997): What Will Be: How the New World of Information Will Change Our Lives. Harper-Collins, New York, NJ, USA.

Dey, A. K., Salber, D., and Abowd, G. (2001): A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human- Computer Interaction, Vol 16.

Fischer, G. (2002): Beyond 'Couch Potatoes': From Consumers to Designers and Active Contributors. Available at http://firstmonday.org/issues/issue7_12/fischer/, FirstMonday (Peer-Reviewed Journal on the Internet), 7 (12), 2002

Fowler, M. (2006): Continuous Integration. http://martinfowler.com/articles/continuousIntegration.html, last update: May, 1st, 2006.

Hydra (2006a): D2.1 Scenarios for usage of HYDRA in 3 different domains. Hydra Project Deliverable, IST project 2005-034891, 2006

Hydra (2006b): D2.5 Initial Requirements Report. Hydra Project Deliverable, IST project 2005-034891, 2006

IEEE 1471 (2000): IEEE Standard 1471-2000: Recommended Practice for Architectural Descriptions of Software-Intensive Systems, IEEE Computer Society.

IEEE 829 (1998): IEEE Standard 829-1998 for Software Test Documentation. Software Engineering Technical Committee of the IEEE Computer Society. IEEE, 1998

ISO (1995): *Open Distributed Processing Reference Model* ISO/IEC IS 10746

Kelly, D., Shepard, T. (2002): Qualitative observations from software code inspection experiments. Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, Toronto, Ontario, Canada.

Krakowiak, S. (2003): http://middleware.objectweb.org/

Leblang, D.B.; Levine, P.H. (1995): Software Configuration Management: Why is it needed and what should it do? In: Estublier, J. (ed.): Software Configuration Management. ICSE SCM-4 and SCM-5 Workshops, pp. 53-60, LNCS, Springer, Berlin.

Lewis, W.E. (2005): Software Testing and Continuous Quality Improvement (2nd Edition). William E. Lewis. Auerbach Publishers

Lieberman, H., Paternó, F. Wulf, V. (eds.) (2006): End User Development. Springer, Berlin, Germany.

Henderson, A., Kyng, M. (1991): There's no Place like Home: Continuing Design in Use. In Greenbaum, J. and Kyng, M. eds. Design at work: Cooperative Design of Computer Systems, Lawrence Erlbaum Ass., Hillsdale, NJ, 1991, pp. 219-240.

Mackus, A., Fielding, R., Herbsleb, J.D. (2002): Two case studies of open source software development: Apache and mozilla, ACM Transactions Software Engineering. In: ACM Transactions on Software Engineering Methodology, 11(3):309-346.

MacLean, A., Carter, K., Lövstrand, L., Moran, T.P. (1990): User-Tailorable Systems: Pressing the Issue with Buttons. In Int. Conference on Computer-Human-Interation (CHI'90), (Seattle, WA. USA, 1990), ACM Press, pp. 175-182.

Mørch, A. (1997): Method and Tools for Tailoring of Object-oriented Applications: An Evolving Artefacts Approach. PhD-Thesis, University of Oslo, Department of Computer Science, Research Report 241, Oslo, Norway.

Ommering, R. v. (2003): Configuration Management in Component Based Product Populations. In: Westfechtel, B. (ed.): Software Configuration Management, Springer.

Pressman, R.S. (2005): Software Engineering: A Practitioner's Approach. McGraw-Hill Professional, ISBN 0072853182, p. 880

Reuter, J.; Hänssgen S.U.; Hunt, J.J.; Tichy, W.F. (1996): Distributed Revision Control Via the World Wide Web. In: Sommerville, I. (ed.): Proceedings of the Sixth International Workshop on Software Configuration Management (SCM-6) in conjunction with the 18th International Conference on Software Engineering (ICSE), p. 166-174, Springer Verlag, Berlin, Germany.

Rozanski, N. and Woods, E. (2005): Software systems architecture: working with stakeholders using viewpoints and perspectives, Pearson Education.

Schmidt, D.C. (2002): Middleware for real-time and embedded systems. Communications of the ACM, 45 (6): pp. 43-48

Spinellis, D. (2003): Code Reading – The Open Source Perspective. p. 528, Addison Wesley, ISBN: 0201799405.

Sutter, H., Alexandrescu, A. (2004): C++ Coding Standards – 101 Rules, Guidelines, and Best Practices. Addison-Wesley Longman, Amsterdam, ISBN: 0321113586, p. 240

Tanenbaum, A.S. and Van Steen, M.: (2007) Distributed Systems. Principles and Paradigms. Addison-Wesley

Thomas, D.; Hunt A. (2004): Pragmatisch programmieren – Versionsverwaltung mit CVS. Hanser Fachbuchverlag, ISBN: 3446228268, p.184

Westfechtel, B. (1996): Document Type Independent Tools: Common Services for Manipulation, Layout and User Support. In: Nagl, M. (ed.): Building Tightly Integrated Software Development Environments: The IPSEN Approach. Springer, New York, NY, USA, pp. 208-221.

Weischedel, G.; Versteegen, G. (2002): Konfigurationsmanagement. Springer.

Westfechtel, B.; Conradi, R. (2003): Software Architecture and Software Configuration management. In: Westfechtel, B.; Hoek, A. v. d. (eds.); Software Configuration Management. Springer

Wulf, V., & Golombek B. (2001): Direct Activation: A Concept to Encourage Tailoring Activities. Behaviour and Information Technology, 20 (4), pp. 249-263.