


# Contract No. IST 2005-034891

# Hydra

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

# D12.2 External developers workshops teaching material

**Integrated Project**
**SO 2.5.3 Embedded systems**

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

**Published by the Hydra Consortium** **20/12/2007 - version 1.3**
**Coordinating Partner: C International Ltd.**

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

**Dissemination Level: Restricted to Group**

**Document file:** D12.2 External developers workshops teaching material v1.3.doc

**Work package:** WP12 - Training

**Task**: T12.2 - Training
**Document owner:** Atta Badii (University of Reading)

**Document history:**

| Version | Author(s) | Date | Changes made |
|---|---|---|---|
| 1.0 | Atta, Badii, Renjith Nair, Daniel Thiemert, Adedayo Adetoye | 31-03-2007 | First Version of the document |
| 1.1 | Members of WP4, WP5, WP6, and WP7 | 14-12-2007 | Functions and class descriptions |
| 1.2 | Atta, Badii, Renjith Nair, Daniel Thiemert, Adedayo Adetoye | 17-12-2007 | Function definitions, Organising and Formatting the inputs |
| 1.3 | Atta Badii, Daniel Thiemert, Renjith Nair, Adedayo Adetoye | 20-12-2007 | Incorporating Reviewer comments |
| | | 20-12-2007 | Final version submitted to the European Commission |

**Internal review history:**

| Reviewed by | Date | Comments |
|---|---|---|
| Manuel Mattheß (Fraunhofer SIT) | 18-12-2007 | Approved with comments |
| Jesper Thestrup (In-JET) | 19-12-2007 | Approved with comments |

**Index:**

# 1. Introduction

## 1.1 Purpose, context and scope of this deliverable

The training dimension of the Hydra project is essential to guarantee the project long-term impact and hence, several training activities have been planned. Some are directed towards consortium members and offer inside training in use of technology and software tools, while others are directed towards external developers developing embedded software systems.

This document is the 'External Technology Workshops Teaching Material' which forms the deliverable D12.2 as part of the WP12. It gives overview of the entire Hydra architecture. This training document will provide high-level introduction to the Hydra Middleware starting from the overall project aims and objectives and providing various holistic views to show the potential applications and plausible usage.

The aim of this document is to give a better idea of what can be done using the Hydra middleware, and which are the major benefits. Focus will be on the communication process in networked systems and how the various parts of the Hydra middleware support the specific needs of communication involving heterogeneous devices.

This document will help the trainees in understanding:

- What is Hydra – in terms of the overall Framework as well as the individual functionalities?
- Which is the role of Hydra middleware in embedded systems communication?
- How to use Hydra in developing programs for communication involving networked embedded devices?
- How to integrate Hydra with the existing communication scenario of embedded devices?
- Design implications while developing programs using Hydra
- The benefits of adopting Hydra in respect to traditional approach

The Overview document is directed towards the following group of people:

- Individuals interested to have a general overview of Hydra middleware
- Application Developers interested to have a first overview of:
    - Impact of Hydra technology in their solutions
    - Technical innovation of Hydra with respect to the state-of-the-art
- Device Manufacturers interested to know more about Hydra regarding:
    - How Hydra will support their devices?
    - How can the devices be made Hydra compliant?
    - What kind of security Hydra will provide?
- Research, Technology, and Business Development Managers interested to have an overview of the whole architecture and assessing the advantages of using the Hydra framework for:
    - Testing and developing new solutions
    - Promoting and distributing their new solutions for a large audience

# 2. Executive summary

This document is the 'External Technology Workshops Teaching Material' which forms the deliverable D12.2 as part of the WP12.

Training is an essential part of any project as it gives an overall view of the project and shows how the project encompasses the stakeholders' interests. It is essential in professional development of trainees as it enriches them with the new and advanced methods which Hydra can provide to their ambient programming environment.

This training document gives an overview of the Hydra architecture and elucidates functions available for each manager thereby providing the developers and manufacturers a glimpse of what can be achieved through Hydra. The functionalities are hereby divided into Device and Application Elements according to the architecture depicted in figure 1.

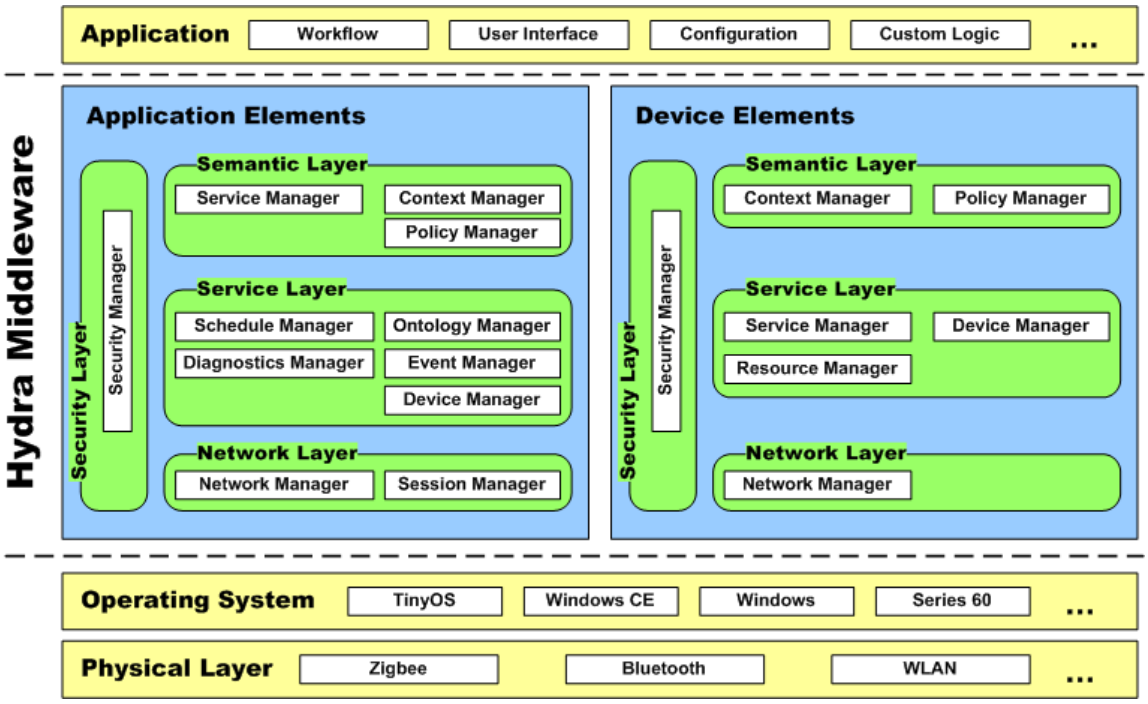


*Figure 1: Hydra Architecture*

However, it must be noted that this deliverable is submitted in month 18 and so it is not the complete Hydra-training document, but a first step towards providing Hydra-training to external developers.

# 3. Hydra Overview

Hydra is a middleware for building secure, fault-tolerant Networked Embedded Systems where diverse heterogeneous devices co-operate to achieve a given goal. It aims to hide the complexity of the underlying infrastructure while providing open interfaces to third parties for application development.

The Hydra project aims to develop middleware based on a Service-oriented Architecture (SOA), to which the underlying communication layer is transparent. The middleware will include support for distributed as well as centralised architectures, security and trust, reflective properties and model-driven development of applications. One of the main advantages of Hydra middleware is that it will be deployable on both new and existing networks of distributed wireless and wired devices, which operate with limited resources in terms of computing power, energy and memory usage. It will allow for secure, trustworthy, and fault tolerant applications through the use of novel distributed security and social trust components and advanced Grid technologies. The embedded and mobile Service-oriented Architecture will provide interoperable access to data, information and knowledge across heterogeneous platforms, including web services, and support true ambient intelligence for ubiquitous networked devices.

One of the major outcomes of the Hydra project will be a SDK (Software Development Kit) which will be used by developers to develop innovative Model-Driven applications with embedded ambient intelligence using the Hydra middleware. Since producers of devices are increasingly facing the need for networking in order to provide higher value-added solutions for their customers, the Hydra project will develop value-oriented business models allowing the producers, in particular Small and Medium-sized Enterprises (SME), to develop solutions that are both technically and commercially viable. The Hydra middleware, SDK toolkit and the business models will be validated in real end-user scenarios in three user domains: Building Automation, Healthcare, and agriculture. Furthermore a Device Development Kit (DDK) will be provided that enables device manufacturers to produce Hydra-enabled devices. Additionally, an Integrated Development Environment (IDE) will be provided. The project also aims to provide an open source reference implementation that will demonstrate the applicability and quality characteristics of the Hydra middleware.

## 3.1 Application Domains of Hydra

To validate the Hydra middleware, three user domains were carefully chosen: Building Automation, Healthcare, and Agriculture.

Building Automation includes Intelligent or Smart Homes and Buildings, and Building Management Systems. This if course includes sensor equipment for monitoring, applications for control and reporting, as well as technologies for automated manufacturing.

Healthcare includes eHealth services such as the electronic patient record or drug and day management as well as smart and intelligent monitoring devices for patient status monitoring.

Agriculture finally entails applications for monitoring of cattle as for automated feeding purposes as well as for example automated control and management of harvesters. Large sensor networks can be deployed in this domain for monitoring which have special requirements and constraints.

# 4. Hydra Architecture

The top level architecture view of Hydra is given in Figure 2. It shows the different managers in the application view (Application Elements) and the device view (Device Elements).

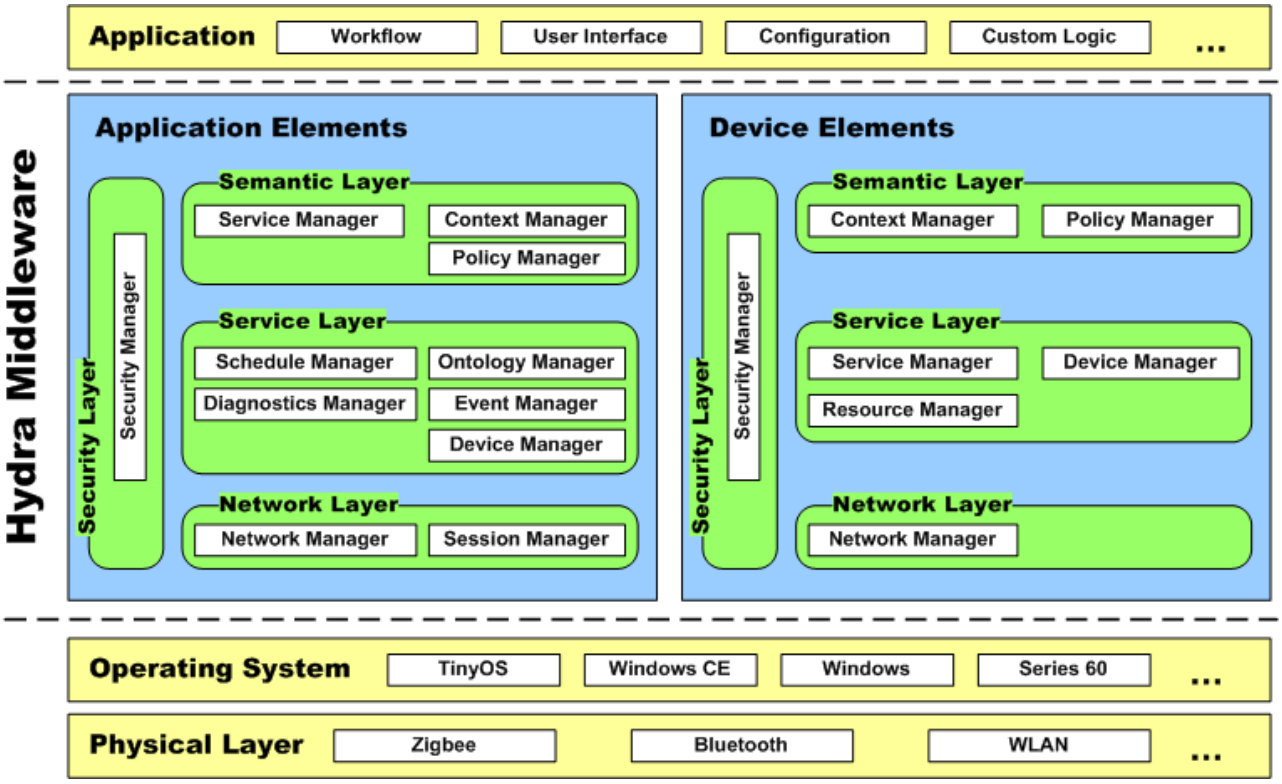


*Figure 2: Hydra Architecture*

The functions for each manager in both application and device elements are explained in the next section.

## 4.2 Device Elements

In the device view, the following managers are primarily involved:

- Service Manager
- Context Manager
- Policy Manager
- Schedule Manager
- Diagnostics Manager
- Ontology Manager
- Event Manager
- Device Manager
- Network Manager
- Session Manager

The communication links between these managers are shown in Figure 3.

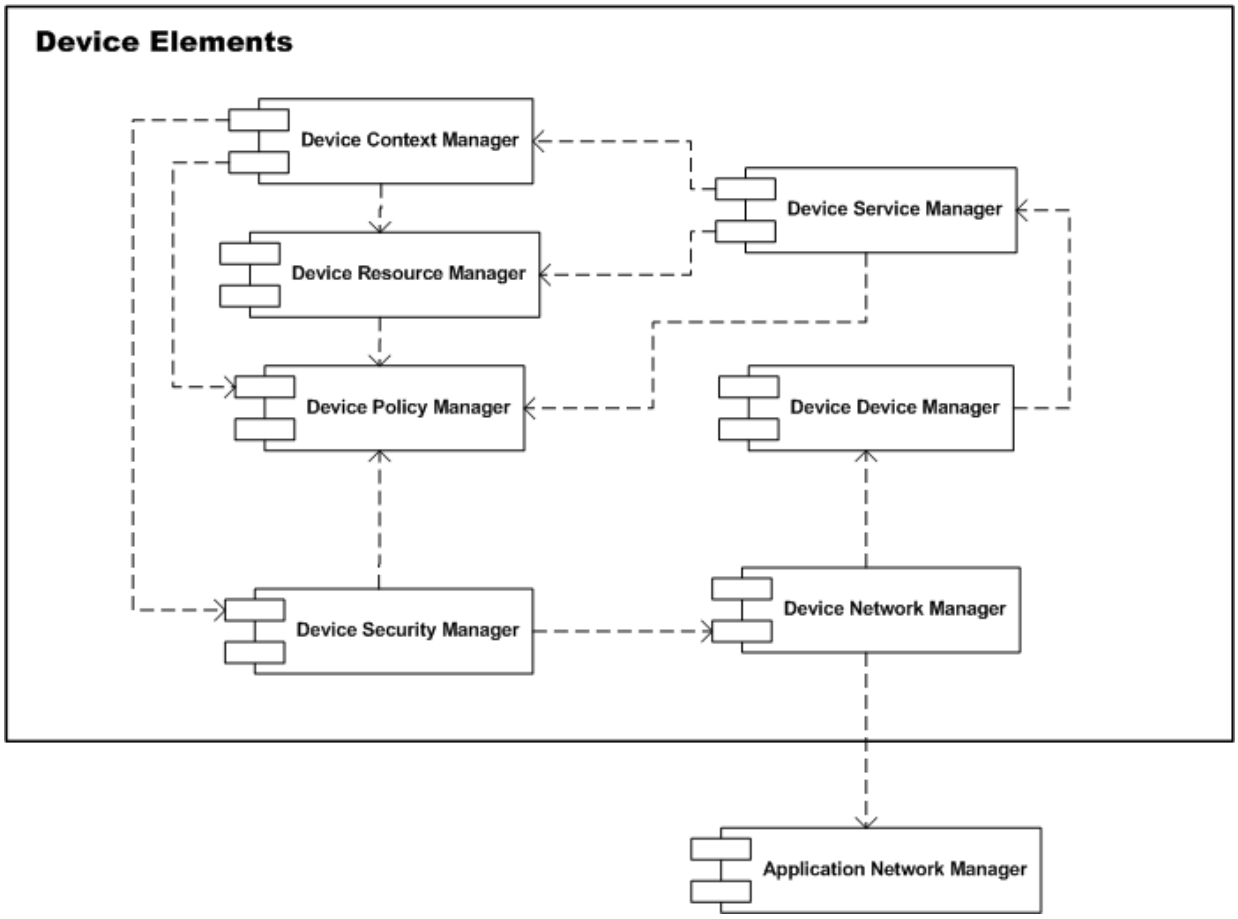


*Figure 3: Device Elements and their communication Links*

### 4.2.1 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."

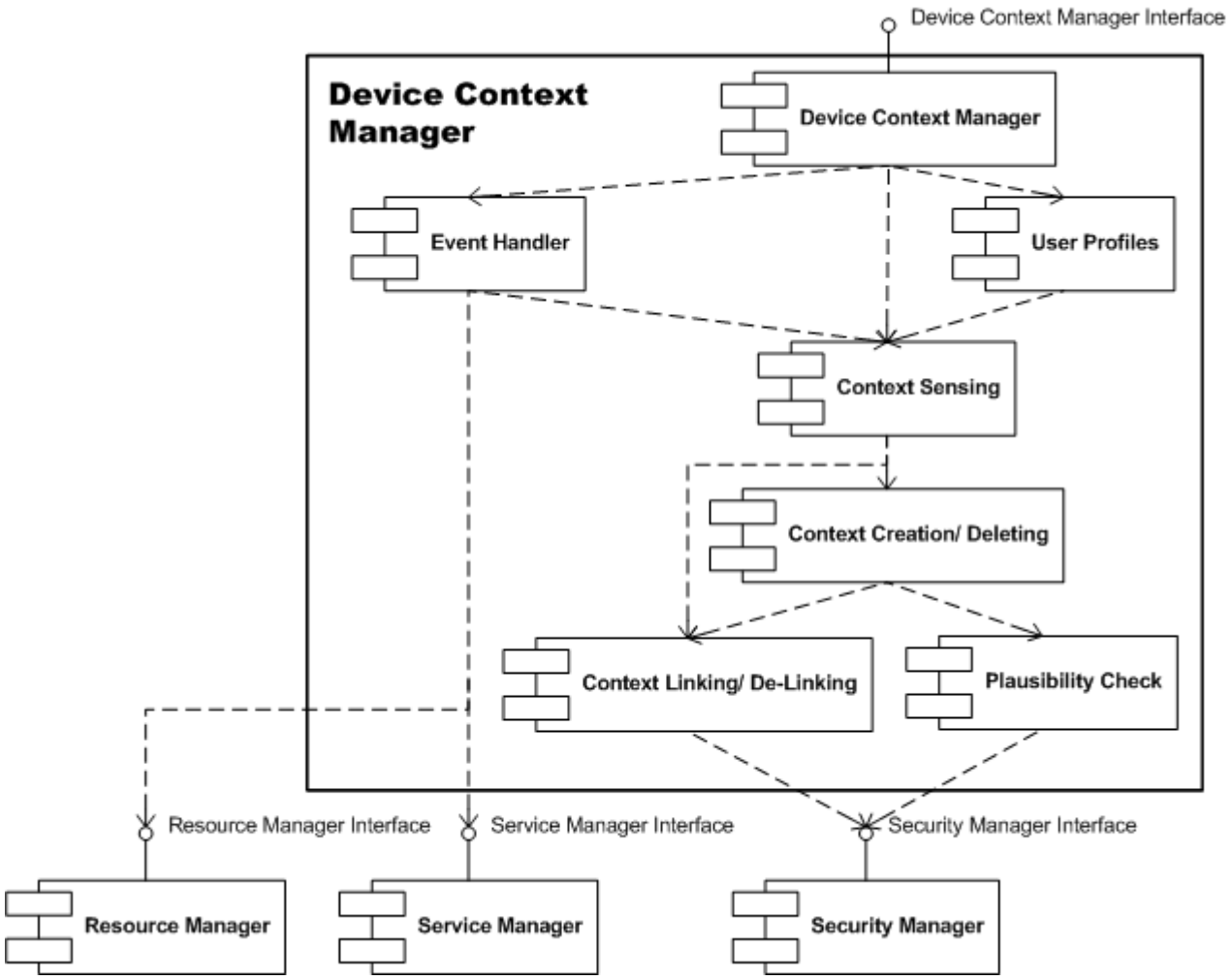


*Figure 4: Device Context Manager*

**Purpose:** The Device Context Manager is 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 realises 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

## Public Functions

| **1. getContext (java.lang.String getContextRequest)** | |
|---|---|
| Returns the current context | |
| **Parameters:** | getContextRequest specifies properties of the context to be returned. |
| **Returns:** | The Hydra Context |

| **2. initiate (String something)** | |
|---|---|
| Starts an action | |
| **Parameters:** | Something Indicates the action to be performed. Example values (which are currently valid and application specific) are serviceRequest , errorConfirmation , errorAnalysis , firmwareDownload and firmwareInstall . |
| **Returns:** | String indicating the status of this function-execution |

| **3. notify (String topic, Part[] partList)** | |
|---|---|
| Notifies the manager about an incoming event<br>This method is called by the EventManager after an event of topic 'topic' has been published. | |
| **Parameters:** | topic: the topic of the event.<br>partList: the event as a key/value array. |
| **Returns:** | True/false depending on whether the function execution was successful or not |

### 4.2.2 Device Device Manager

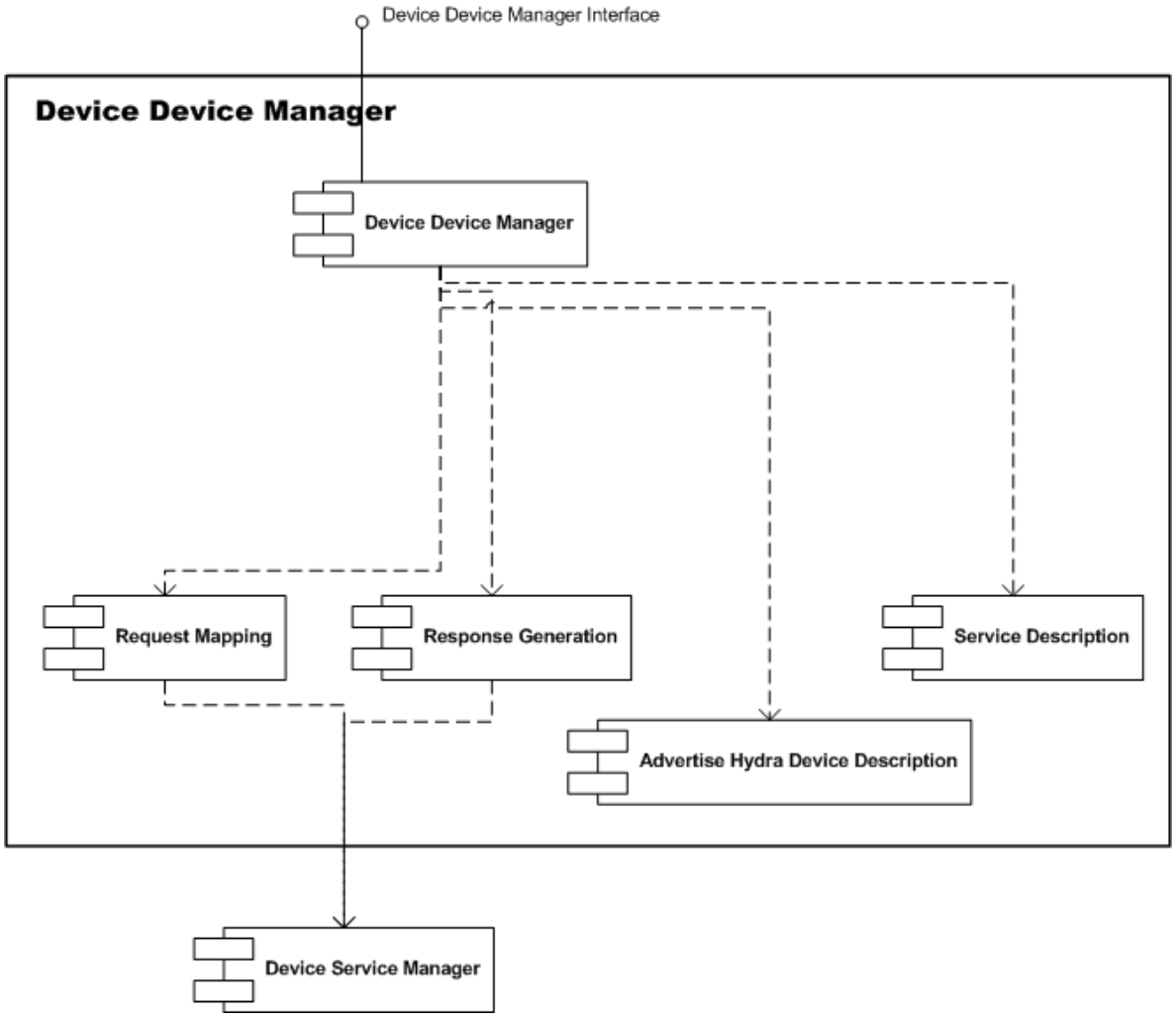


*Figure 5: Device - Device Manager*

**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 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

**Public Functions**

| **1. RegisterError (string property, string errorcode)** | |
|---|---|
| Registers an error condition | |
| **Parameters:** | property: The error property as string.<br>errorcode: The error code as string. |
| **Returns:** | A string containing the registered error |

| **2. SendErrorMessage (string deviceid, string message)** | |
|---|---|
| Sends an error message for a specific device | |
| **Parameters:** | deviceid: The device id as string.<br>message: The error message as string. |
| **Returns:** | A string containing the sent error message |

| **3. ForwardCredentials (string credentials)** | |
|---|---|
| Forwards credentials for this device | |
| **Parameters:** | credentials: The credentials as string. |
| **Returns:** | A string containing the forwarded credentials |

| **4. Install (string firmware, string code)** | |
|---|---|
| Installs firmware for this device | |
| **Parameters:** | firmware: The firmware as string.<br>code: Optional code as string. |
| **Returns:** | A string indicating the status of the firmware installation |

| **5. Execute (string serviceid, string methodName, string parameters, string values)** |
|---|
| Executes a specific method for a service (using the device service manager) |

| | |
|---|---|
| **Parameters:** | serviceid: The serviceid as string.<br>methodName: The methodName as string.<br>parameters: A comma delimited string with the parameter names.<br>parameters: A comma delimited string with the parameter values (Matched against "parameters"). |
| **Returns:** | A string indicating the result of the execute |

| **6. GetStateVariable (string deviceBaseUrl, string serviceid, string variable)** | |
|---|---|
| Retrieves a state variable value for a service (using the device service manager) | |
| **Parameters:** | deviceBaseUrl: Optional string with the deviceBaseUrl.<br>serviceid: The service id as string.<br>variable: A string containing the state variable name. |
| **Returns:** | A string with the state variable value |

| **7. GetDeviceStatus ()** | |
|---|---|
| Retrieves the device status (using the device service manager) | |
| **Parameters:** | |
| **Returns:** | A string with the device status |

### 4.2.3 Device Network Manager

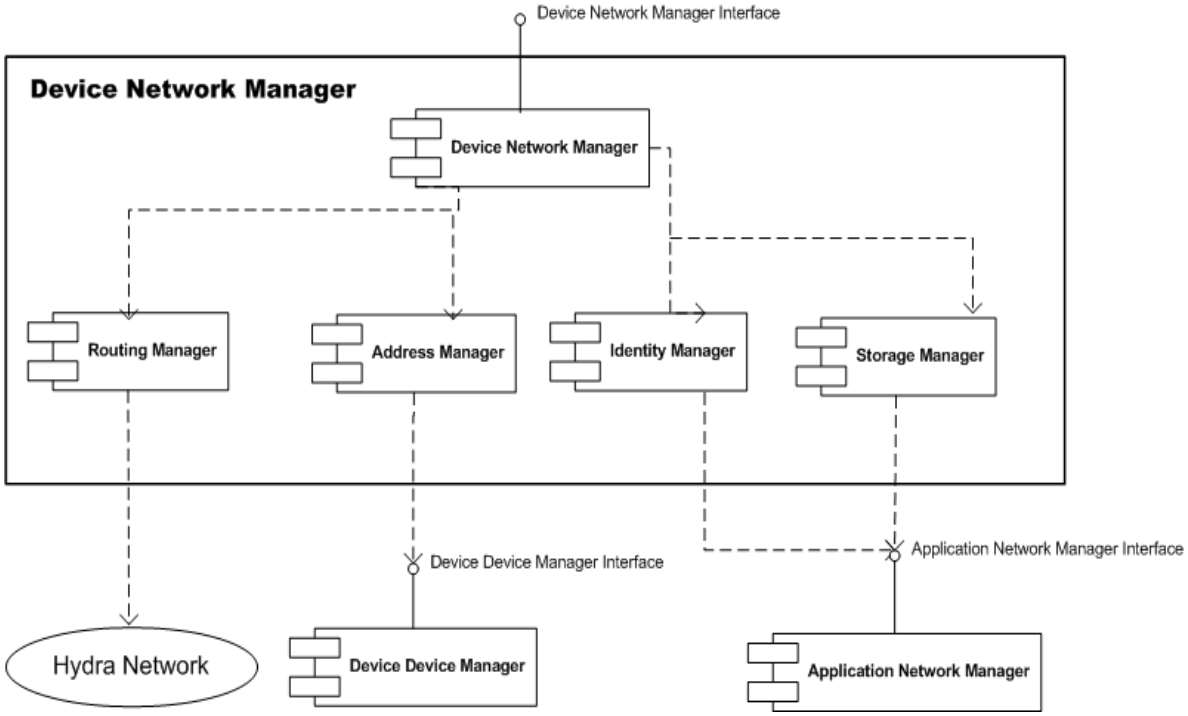


*Figure 6: Device Network Manager*

**Purpose:** The Device Network Manager is responsible for network management on the device. It is needed for routing data from other nodes of the architecture to the devices. This data is sent independently of the communication technology used. It also maintains the Hydra ID (HID) for the devices in its range and the information to establish a connection to them (e.g., the IP addresses for them if they are IP enabled).

It has to be mentioned that in the network architecture proposed in D5.1, we have identified four different levels of devices (not including large machine installation like PCs, servers, etc.) in a network context (further description and classification is made in [6]:

| Name | Description |
|---|---|
| L0 | This device can send data. It could be RFID tags, low power nodes, or other no IP device. Typically it uses a gateway. |
| L1 | This type of device can send and receive data with IP. |
| L2 | This type of device has the same capabilities as L1 devices. Additionally it is capable of routing data to other devices. |
| L3 | L3 devices have the same capabilities as L2 device and are capable of Naming and Location service. |

Although, from the network point of view, we decided to use IP communication, L0 devices do not implement the IP layer, as they need L2 or L3 devices acting as gateways for these small devices. Thus, the figure above represents the network manager in the device view for the higher class devices. Lower ones will not implement all of the subcomponents.

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

**Routing Manager is** responsible for forwarding the data to the appropriate node in the network and for detecting when data has to be consumed in the device itself.

**Identity Manager is** in charge of managing the Hydra ID for the devices it controls from the network point of view. It assures that all HIDs are unique under their scope

(this will happen, for example when a L3 device controls a number of L0 devices). The HID must be unique because it is used to identify devices, applications, users, etc.

**Session Manager** maintains the communication sessions.

**Time Manager is an** internal Manager for time synchronisation.

**Backbone Manager is an** internal Manager which deals mainly with the device to device communication (no external access).

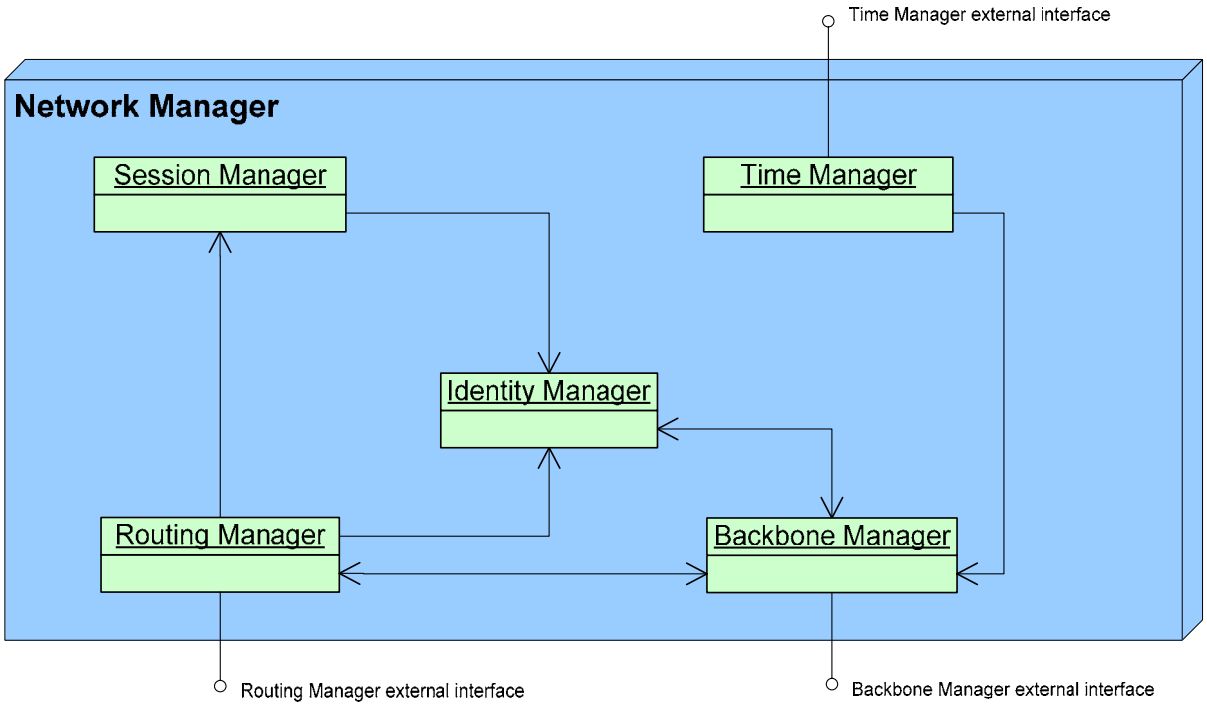


**Figure 1: Sub Managers of network manager**

**Main Functions:**

- Route data
- Send data to the network
- Receive data from the network
- Get an IP address from the network
- Get a Hydra ID
- Maintain shared memory space

**Dependencies:** Device Device Manager, Application Network Manager

## Public Functions of Routing Manager

The Routing Manager will be in charge of sending and receiving the messages to and from the network. At the Network Manager level, it is the single point of communication with the other devices. Its public functions are given below:

| **1. sendData(long sessionID, long senderHID, long receiverHID, String data)** |
|---|
| Operation to send some data to another network manager. It calls the sendData method of the RouteManager. |

| | |
|---|---|
| **Parameters:** | sessionID: The ID of the session related to this communication<br>senderHID: The HID of the sender<br>receiverHID: The HID of the receiver<br>data: The data to send in a String format |
| **Returns:** | An indicator indicating the success of the transmission: 0 = Success, -1 = Echec |

| **2. receiveData(long sessionID, long senderHID, long receiverHID, String data)** | |
|---|---|
| Operation to receive some data from another network manager. It calls the receiveData method of the RouteManager. | |
| **Parameters:** | sessionID: The ID of the session related to this communication<br>senderHID: The HID of the sender<br>receiverHID: The HID of the receiver<br>data: The data to send in a String format |
| **Returns:** | An indicator indicating the success of the transmission: 0 = Success, -1 = Echec |

## Public Functions of Session Manager

| **8. openSession(long senderHID, long receiverHID)** | |
|---|---|
| Operation to open a session between two HIDs. It calls the createSession method of the SessionManager. | |
| **Parameters:** | senderHID: The HID of the source node (client)<br>receiverHID: The HID of the destination node (server) |
| **Returns:** | The SessionID generated for this session |

| **9. closeSession(long sessionID)** | |
|---|---|
| Operation to close a session. It calls the closeSession method of the SessionManager. | |
| **Parameters:** | sessionID: The sessionID of the session to be closed |
| **Returns:** | - |

| **10. getSessionParameter(long sessionID, String key)** | |
|---|---|
| Operation to request the value of a given session data. It calls the getSessionParameter method of the SessionManager. | |
| **Parameters:** | sessionID: The sessionID of the session to be asked<br>key: The requested parameter |

| | |
|---|---|
| **Returns:** | The value of the requested parameter |

| **11. setSessionParameter(long sessionID, String key, String value)** | |
|---|---|
| Operation to set the value of a given session data. It calls the setSessionParameter method of the SessionManager. | |
| **Parameters:** | sessionID: The sessionID of the session to be asked<br>key: The requested parameter<br>value: The new value of the requested parameter |
| **Returns:** | - |

| **12. synchronizeSessionsList(long senderHID)** | |
|---|---|
| Operation to synchronize the clientSessionsList with the serverSessionsList of this host. It calls the synchronizeSessionsList method of the SessionManager. | |
| **Parameters:** | senderHID: The HID of the client that need to synchronize its clientSessionsList with the serverSessionsList of this host |
| **Returns:** | A vector that contains the sessionID to be deleted |

## Public Functions of Identity Manager

The objective of the Identity Manager is to provide a unique identifier for each entity of the Hydra network. Externally accessible functions of Identity Manager are given below:

| **1. createHID()** | |
|---|---|
| Generates a context free HID | |
| **Parameters:** | - |
| **Returns:** | A new context free HID |

| **2. createHID(contextID, level)** | |
|---|---|
| Generates an HID in a given context | |
| **Parameters:** | contextID: ID of the context that will be linked to this HID<br>level: Number of nested contexts |
| **Returns:** | A new HID with the following structure:<br>context(n)ID.context(n-1)ID....context(0)ID.deviceID<br>where<br>deviceID is a random ID<br>and |

| | |
|---|---|
| | context(n)ID is a random ID if n < level, where 0 <= n <= 3.<br>If n = level then context(n)ID is equal to the contextID input parameter. |

| **3. renewHID ()** | |
|---|---|
| Renews a context free HID | |
| **Parameters:** | - |
| **Returns:** | A new context free HID |

| **4. createHID(contextID, level)** | |
|---|---|
| Renews an HID in a given context | |
| **Parameters:** | contextID: ID of the context that will be linked to this HID<br>level: Number of nested contexts |
| **Returns:** | A new HID with the following structure:<br>context(n)ID.context(n-1)ID....context(0)ID.deviceID<br>where<br>deviceID is a random ID<br>and<br>context(n)ID is a random ID if n < level, where 0 <= n <= 3.<br>If n = level then context(n)ID is equal to the contextID input parameter. |

| **5. deleteHID(hid)** | |
|---|---|
| Deletes an HID from the Identity Manager IdTable | |
| **Parameters:** | hid: The HID to be deleted |
| **Returns:** | - |

| **6. deleteAllHIDs(hid)** | |
|---|---|
| Delete every HID from the Identity Manager's IdTable | |
| **Parameters:** | - |
| **Returns:** | - |

## Public functions of Storage Manager

The Storage Manager is responsible for storage management on the device level. It is used to provide a single, consistent interface to the storage on the devices. Depending on the device capabilities and application requirements, 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. Apart from easing application IO operations, the general IO access through a local Storage Manager will also allow increased IO performance for all applications, by adding transparent caching. Public functions for the storage manager are given below:

| **1. setFileLocation(String securityContext, String location)** | |
|---|---|
| Sets up storage location by mapping the provided security context to the corresponding HID. This is used to decide the outcome of openFile operations called with the same security context. | |
| **Parameters:** | securityContext: The securityContext to be linked to this file<br>location: The pathfile for this file |
| **Returns:** | - |

| **2. getFileLocation(String securityContext)** | |
|---|---|
| Looks up the HID of the storage location for the given security context | |
| **Parameters:** | securityContext: The securityContext linked to this file |
| **Returns:** | The pathfile for this file |

| **3. openFile(String securityContext, String filename, String mode)** | |
|---|---|
| Opens a file access session given a security context, a filename and an access mode. Returns a file descriptor used for all further operations in the file access session. | |
| **Parameters:** | securityContext: The securityContext linked to this file<br>filename: The name of the file to be opened<br>mode: The opening mode to be used |
| **Returns:** | The FileDescriptor object associated to this file |

| **4. seekFile(FileDescriptor fileHandle, long position)** |
|---|
| Moves the file-pointer offset, measured from the beginning of the file, i.e., moves the point from which the next read or write occurs. |

| | |
|---|---|
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file<br>position: The new position for the file-pointer offset |
| **Returns:** | - |

| **5. tellFile(FileDescriptor fileHandle)** | |
|---|---|
| Provides the current file-pointer offset from the beginning of the file. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file |
| **Returns:** | The current position of the file-pointer offset |

| **6. readFile(FileDescriptor fileHandle, byte[] buffer)** | |
|---|---|
| Reads a specified number of bytes from the file, starting from the current file-pointer. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file<br>buffer: The array where the bytes read will be stored |
| **Returns:** | The number of bytes read |

| **7. readLineFile(FileDescriptor fileHandle)** | |
|---|---|
| Similar read method that reads until the next newline marker. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file |
| **Returns:** | A String containing the line read |

| **8. readStringFile(FileDescriptor fileHandle)** | |
|---|---|
| Similar read method that reads and formats to a string instead of pure bytes. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file |
| **Returns:** | A String containing the formatted file content |

| **9. writeFile(FileDescriptor fileHandle, byte[] buffer)** |
|---|
| Writes provided bytes to the file, starting from the current file-pointer. |

| | |
|---|---|
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file<br>buffer: The bytes to be written to this file |
| **Returns:** | - |

| **10. writeLineFile(FileDescriptor fileHandle, String line)** | |
|---|---|
| Similar write method to write a line to the file. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file<br>line: The line to be written to this file |
| **Returns:** | - |

| **11. writeStringFile(FileDescriptor fileHandle, String line)** | |
|---|---|
| Similar write method to write a string rather than raw bytes to a file. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file<br>line: The String to be written to this file |
| **Returns:** | - |

| **12. getLengthFile(FileDescriptor fileHandle)** | |
|---|---|
| Provides the current length of the file, i.e., the size in bytes. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file |
| **Returns:** | The length of the file |

| **13. setLengthFile(FileDescriptor fileHandle, long length)** | |
|---|---|
| Sets the current length of the file, i.e., the size in bytes. This works as truncate if the file was previously longer than the length specified. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file<br>length: The length to be set for this file |
| **Returns:** | - |

| **14. lockFile(FileDescriptor fileHandle, long position, long length)** |
|---|
| Locks a specified part of a file for shared reading or exclusive writing. The part is specified |

| with the provided offset and length variables. | |
|---|---|
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file<br>position: The offset position indicating where locking will start<br>length: The number of bytes to be locked |
| **Returns:** | - |

| **15. releaseFile(FileDescriptor fileHandle, long position, long length)** | |
|---|---|
| Releases the lock on a specified part of a file. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file<br>position: The offset position indicating where releasing will start<br>length: The number of bytes to be released |
| **Returns:** | - |

| **16. closeFile (FileDescriptor fileHandle)** | |
|---|---|
| Closes the file access session. This includes writing any cached data and revoking all further access to the file using the provided file descriptor. | |
| **Parameters:** | fileHandle: The FileDescriptor object associated to this file |
| **Returns:** | - |

### 4.2.4 Device Policy Manager

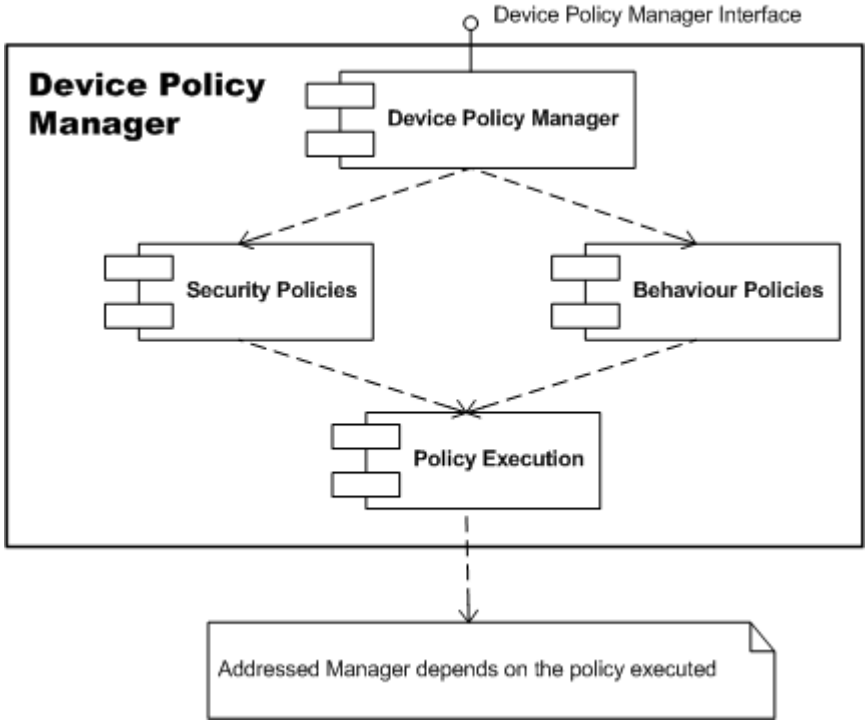


*Figure 7: Device Policy Manager*

**Purpose:** The Device Policy Manager is 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.

## Public functions

| **1. getProtectionProfile (String contextInformation)** | |
|---|---|
| Function to retrieve the protection profile | |
| **Parameters:** | contextInformation: current context information |
| **Returns:** | Protection Information requested (of data type Protection Information) |

Other functions for the policy manager in the device view will be decided and implemented only after the second Iteration.

### 4.2.5 Device Resource Manager

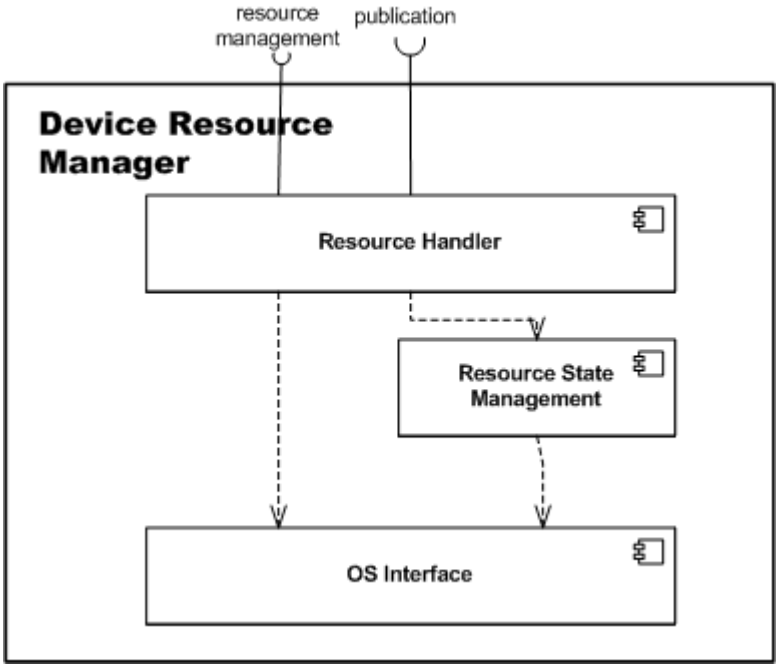


*Figure 8: Device Resource Manager*

**Purpose:** The Device Resource Manager is responsible for 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

## Public Functions

Public functions for the Resource manager will be defined and implemented only after the second iteration.

### 4.2.6 Device Security Manager

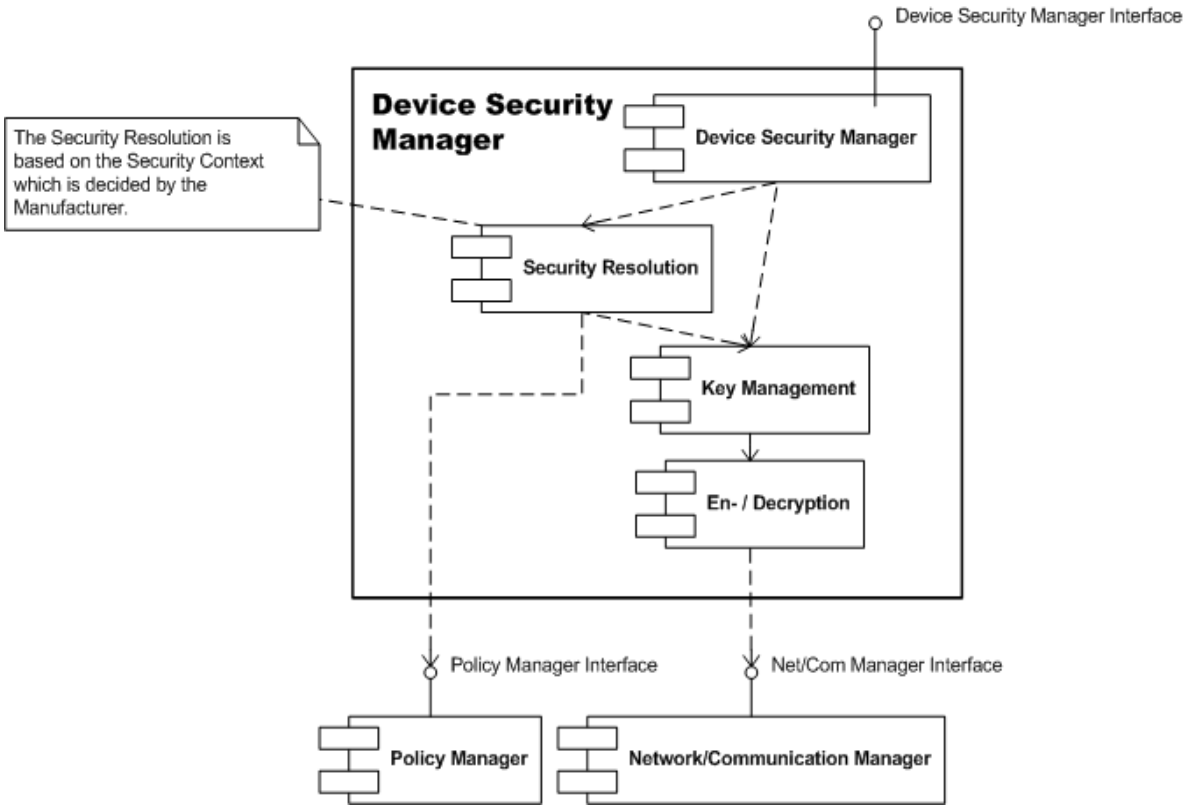


*Figure 9: Device Security Manager*

**Purpose:** The Device Security Manager realises 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

## Public functions

| **1. createKey (java.lang.String identityID, String keyParameters)** | |
|---|---|
| Function to create a secure key for an identity. | |
| **Parameters:** | identityID: Identity ID in string format<br>keyParameters: Key parameters for the key, in string format, separated by comma. |

| | |
|---|---|
| **Returns:** | Unique identifier of the created key or *null* in case of failure. |

| **2. revokeKey (String keyID)** | |
|---|---|
| Function to revoke an already existing key. | |
| **Parameters:** | keyID: The ID of the key to be revoked |
| **Returns:** | String indicating the status of the execution of function |

| **3. deleteKey (String keyID)** | |
|---|---|
| Function to delete a key. | |
| **Parameters:** | keyID: The ID of the key to be deleted |
| **Returns:** | String indicating the status of the execution of function |

| **4. send(String data, Context currentContext, String destinationHID)** | |
|---|---|
| Encrypts and/or signs outgoing data according to the current protection profile and forwards protected data to the NetworkManager. | |
| **Parameters:** | data: Plaintext data to be protected.<br>currentContext: The current context |
| **Returns:** | Status code: -1 means an error occurred during encryption/signing, 0 means success. |

| **5. validate (String something, String credentialID, Context context)** | |
|---|---|
| Function to check if a credential is valid for a certain request. | |
| **Parameters:** | something: request to be validated<br>credentialed: Credential ID, in string format<br>context Context: in which the data is to be validated |
| **Returns:** | True if the validation is successful, else false |

### 4.2.7 Device Service Manager

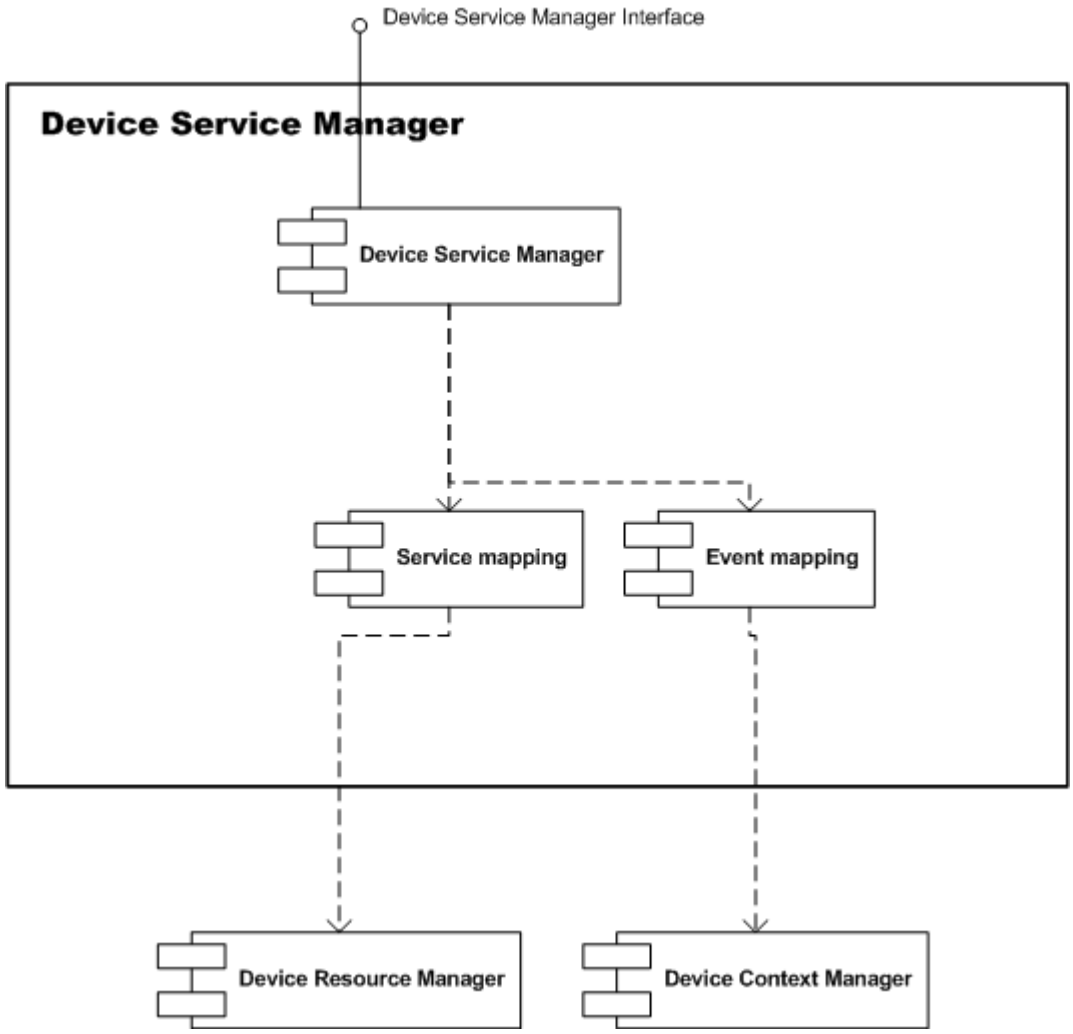


*Figure 10: Device Service Manager*

**Purpose:** The Device Service Manager implements a 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 requests to internal device operations. One device can have several service mappings.

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

**Dependencies:** Device Resource Manager, Device Context Manager

## Public Functions

| 1. | **RegisterError (string property, string errorcode)** |
|---|---|

| Registers an error condition | |
|---|---|
| **Parameters:** | property: The error property as string.<br>errorcode: The error code as string. |
| **Returns:** | A string containing the error to be registered |

| **2. SendErrorMessage (string message)** | |
|---|---|
| Sends an error message for a specific device | |
| **Parameters:** | deviceid: The device ID as string.<br>message: The error message as string. |
| **Returns:** | A string containing the error to be sent |

| **3. ForwardCredentials (string credentials)** | |
|---|---|
| Forwards credentials for this device service | |
| **Parameters:** | credentials: The credentials as string. |
| **Returns:** | A string containing the credentials to be forwarded |

| **4. GetServiceDescription (string serviceid)** | |
|---|---|
| Retrieves a service description for a service | |
| **Parameters:** | serviceid: The service ID as string. |
| **Returns:** | An XmlDocument with service description |

| **5. Execute (string serviceid, string methodName, string parameters, string values)** | |
|---|---|
| Retrieves a service description for a service | |
| **Parameters:** | serviceid: The service ID as string.<br>methodName: The method name as string.<br>parameters: A comma delimited string with the parameter names.<br>values: A comma delimited string with the parameter values (matched against "parameters"). |
| **Returns:** | A string indicating the result of the execution |

| **6. GetStateVariable (string deviceBaseUrl, string serviceid, string variable)** | |
|---|---|
| Retrieves a state variable value for a service | |
| **Parameters:** | deviceBaseUrl Optional string with device's base URL.<br>serviceid The service ID as string.<br>variable A string containing the state variable name. |

| Returns: | A string with the state variable value |
|---|---|

## 4.3 Application Elements

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

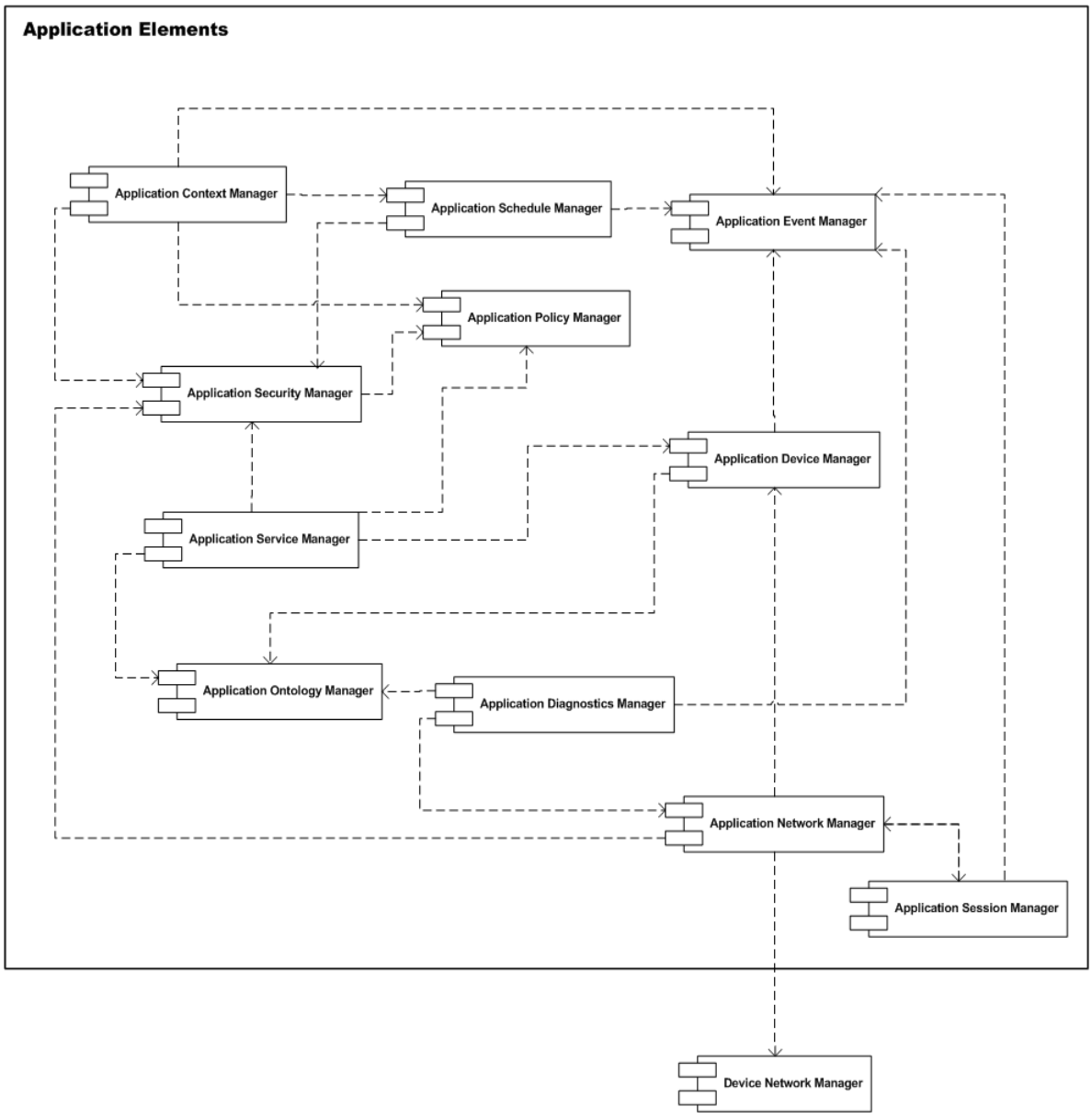


*Figure 11: Overview of Application Elements*

### 4.3.1 Application Context Manager

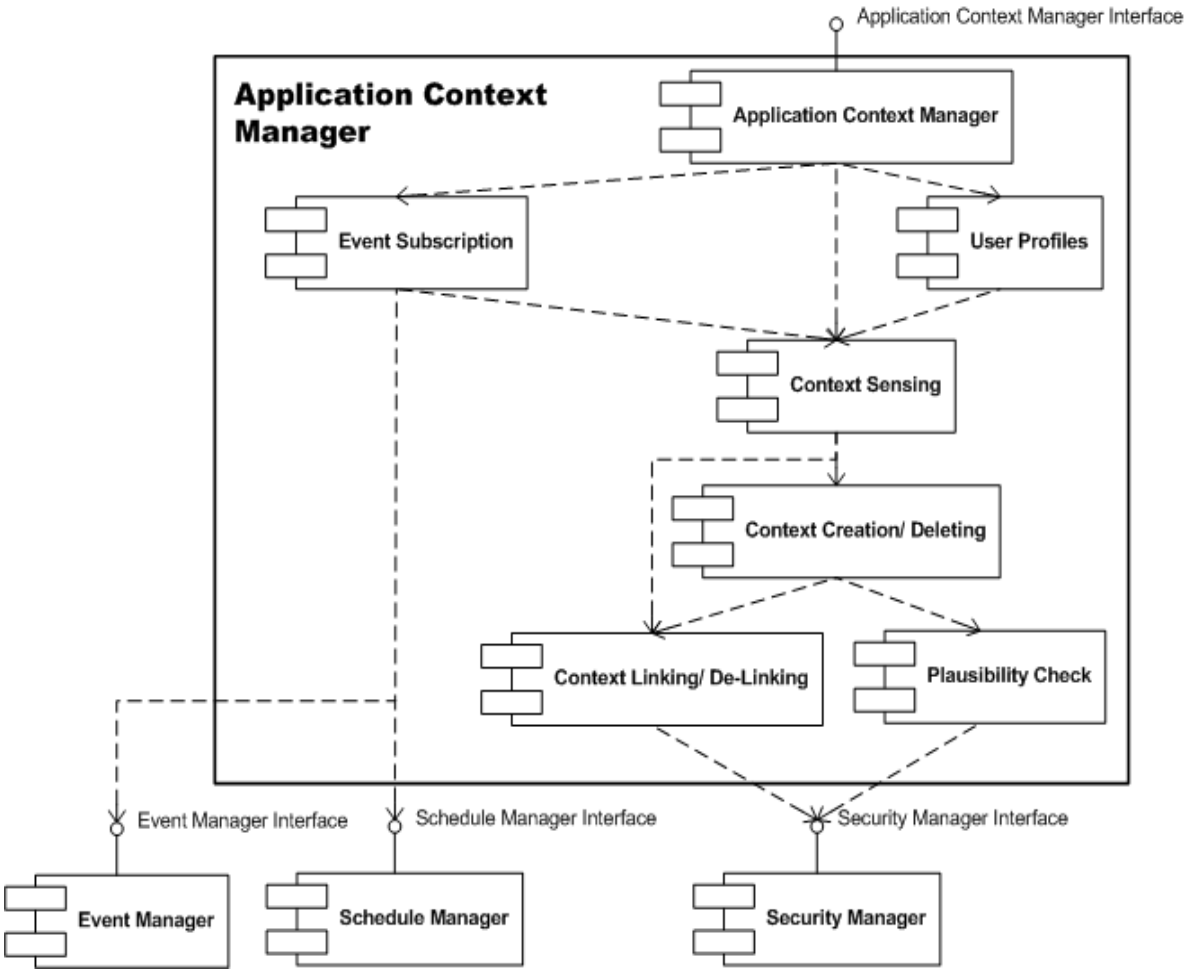


*Figure 12: Application Context Manager*

**Purpose:** The Application Context Manager is responsible for realisation of context for the application elements.

**Main Functions:**

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

**Description:** In order to establish a semantic framework as shown in the figure below, the Application Context Manager is needed. It realises 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 devices 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. Additionally, 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

**Public functions**

Public functions for Application context manager are the same as the ones available in device context manager. So please refer to Device Context Manager for the function-descriptions.

### 4.3.2 Application Device Manager

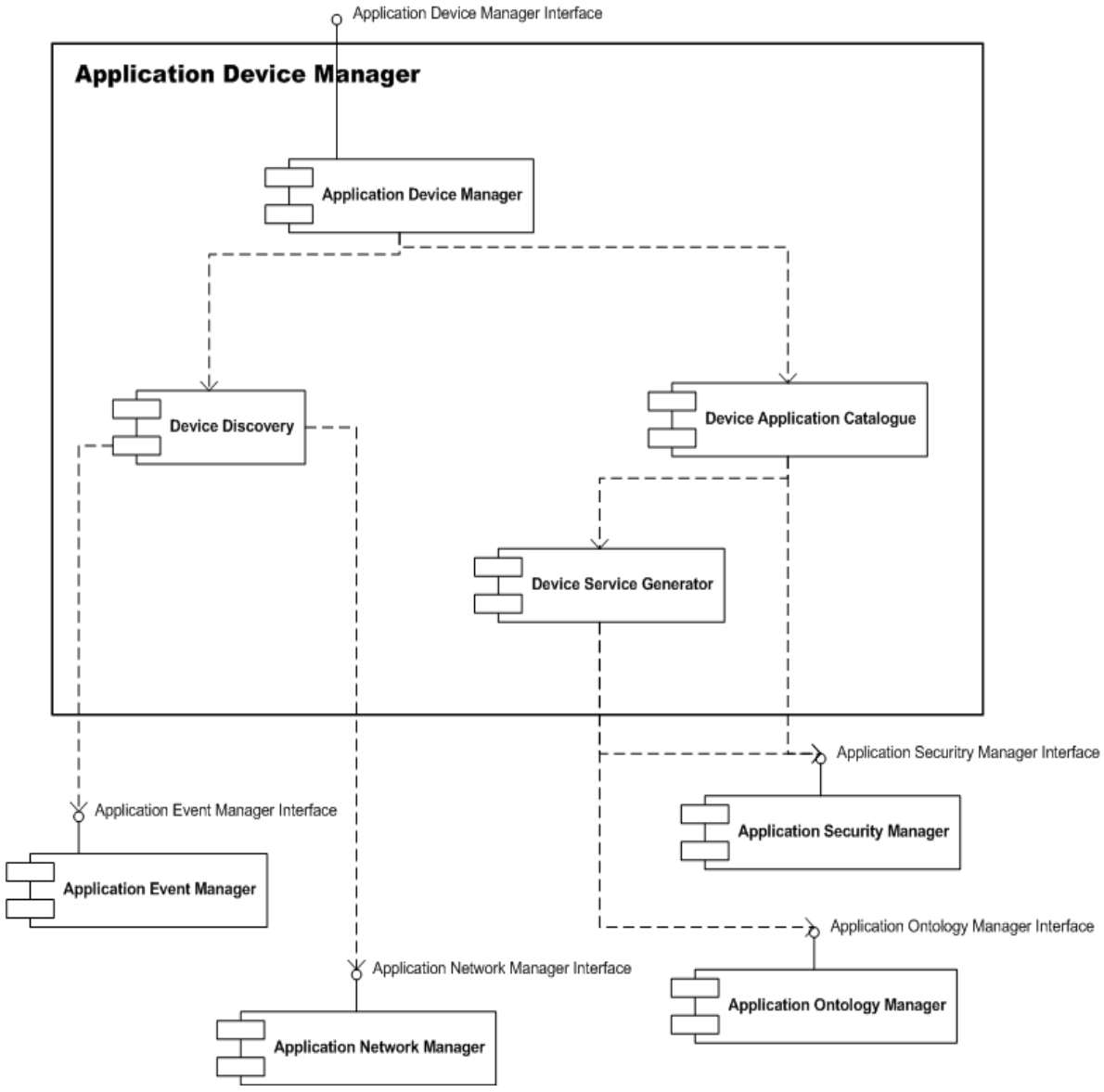


*Figure 13: Application Device Manager*

**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 virtualisation
- 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 different discovery protocols, at least UPnP (Universal Plug and Play). It will also support user-initiated discovery schemes.

**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. **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.

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

### Public Member Functions

| **1. ProcessErrorMessage (XmlNode theMessage)** | |
|---|---|
| Processes an error message | |
| **Parameters:** | theMessage: The error message as an XML Node. |
| **Returns:** | Description of the error |

| **2. ProcessErrorMessageString (string theMessage)** | |
|---|---|
| Processes an error message given in string format. | |
| **Parameters:** | theMessage: The error message as a string. |
| **Returns:** | Description of the error |

| **3. SetDeviceStatus (string deviceid, string statusmessage)** | |
|---|---|
| Sets the device status | |
| **Parameters:** | deviceid: The ID of the device.<br>statusmessage A message describing the changes to be made to the device status. |
| **Returns:** | The updated device status |

| 4. GetDeviceInfo (string deviceid) | |
|---|---|
| Returns status and other information of the device | |
| **Parameters:** | deviceid: The ID of the device. |
| **Returns:** | The current device status and information |

| 5. GetDeviceStatus (string deviceid) | |
|---|---|
| Returns the device status as an XML Node | |
| **Parameters:** | deviceid: The ID of the device. |
| **Returns:** | The current device status |

| 6. GetDeviceXML (string deviceid) | |
|---|---|
| Returns the XML description of the device | |
| **Parameters:** | deviceid: The ID of the device. |
| **Returns:** | An XML Node containing the description of the device. |

| 7. GetDevices (string type) | |
|---|---|
| Returns a list of the devices currently available in the network | |
| **Parameters:** | type: A device type. |
| **Returns:** | A list of the currently available devices |

| 8. GetDevicesAsXML (string type) | |
|---|---|
| Returns a list of the devices currently available in the network | |
| **Parameters:** | type: A device type. |
| **Returns:** | An XML Document containing a list of the currently available devices |

| 9. GetDeviceOntologyDescriptionAsXML (string deviceontology_id) |
|---|
| Returns the ontology description of the device as an XML Document |

| | |
|---|---|
| **Parameters:** | deviceontology_id: The ID of the device. |
| **Returns:** | The ontology description of the device as an XML Document |

| **10. GetDeviceOntologyDescription (string deviceontology_id)** | |
|---|---|
| Returns the ontology description of the device as a string | |
| **Parameters:** | deviceontology_id: The ID of the device. |
| **Returns:** | The ontology description of the device as a string |

| **11. GetProperty (string deviceid, string property)** | |
|---|---|
| Returns the named property of the device | |
| **Parameters:** | deviceid: The ID of the device.<br>property: The name of the property. |
| **Returns:** | The value of the property |

| **12. HasProperty (string deviceid, string property)** | |
|---|---|
| Indicates if the device has a property with the specified name | |
| **Parameters:** | deviceid: The ID of the device.<br>property: The name of the property. |
| **Returns:** | True if the property exists, false otherwise. |

| **13. SetProperty (string deviceid, string property, string value)** | |
|---|---|
| Sets the named property of the device | |
| **Parameters:** | deviceid: The ID of the device.<br>property: The name of the property.<br>value: The new value of the property. |
| **Returns:** | The new value of the property |

| **14. Execute (string deviceid, string serviceid, string methodName, string parameters, string values)** |
|---|
| Generic method to execute any method in a service on a device |

| | |
|---|---|
| **Parameters:** | deviceid: The ID of the device.<br>serviceid: The ID of the service.<br>methodName: The name of the method.<br>parameters The parameters of the method.<br>values: The values of the parameters. |
| **Returns:** | The result of execution of the method |

| **15. Install (string firmware, string code, string deviceid)** | |
|---|---|
| Will install new firmware to a specified device | |
| **Parameters:** | firmware: The URI to download the firmware.<br>code: The code to unlock the device.<br>deviceid: The ID of the device. |
| **Returns:** | |

### 4.3.3 Application Diagnostics Manager

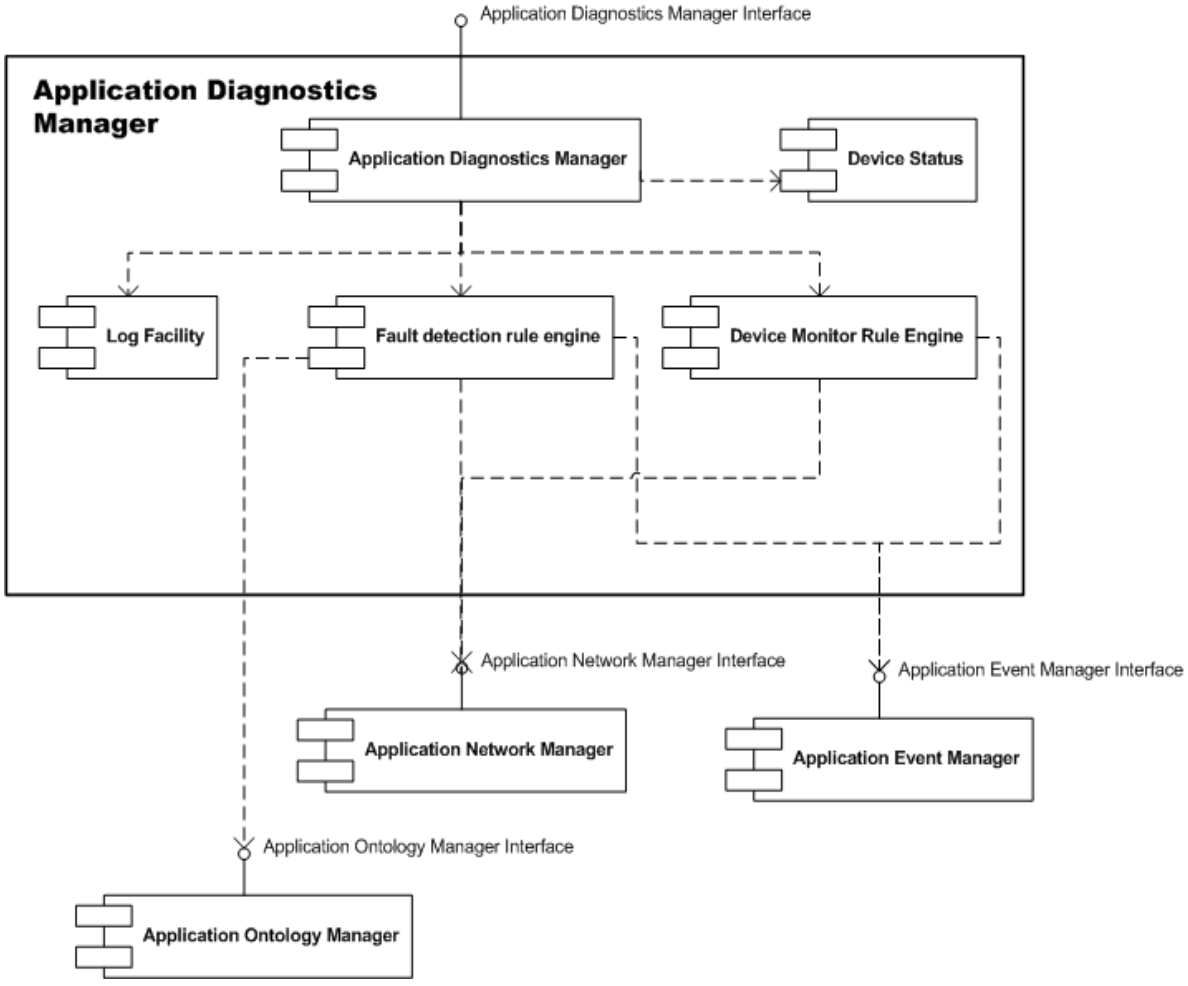


*Figure 14: Application Diagnostics Manager*

**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/ does not 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.

**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 Rule Engine:** This rule engine applies a rule set to discover if there is any malfunctioning or strange behaviour in the system.

**Device Monitor Rule Engine:** This rule engine applies a rule set to monitor devices in order to be preemptive to avoid errors and malfunctions, for instance by monitoring the resource usage of certain devices. This is done by using the information in the Device Ontology.

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

## Public Functions

Public functions for the diagnostics manager will be defined and implemented only after the 2nd Iteration.

### 4.3.4 Application Event Manager

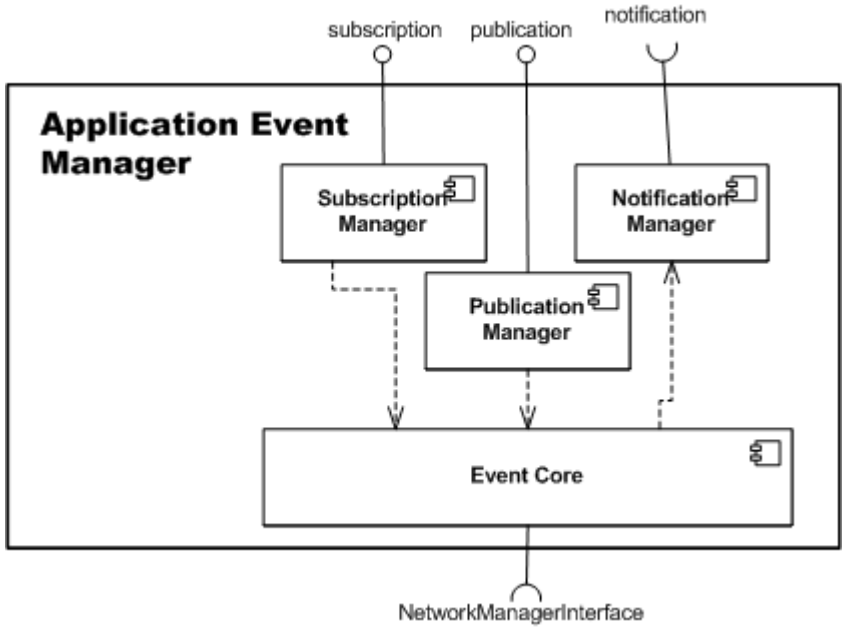


*Figure 15: Application Event Manager*

**Purpose:** The Application Event Manager is 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 receiver do not need to be online 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 used 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

## Public Functions

| **1. subscribe (java.lang.String topic, org.apache.axis.types.URI subscriber)** | |
|---|---|
| Subscribe method - Saves a subscription made by subscriber with the given URI, to the specific topic. | |
| **Parameters:** | topic: topic to be subscribed<br>subscriber: address of the subscriber |

| | |
|---|---|
| **Returns:** | true or false depending on the result of saving the subscription |

| **2. unsubscribe (java.lang.String topic, org.apache.axis.types.URI subscriber)** | |
|---|---|
| Unsubscribe method - Deletes a subscription made by subscriber with the given URI, to the specific topic. | |
| **Parameters:** | topic: topic to be subscribed<br>subscriber: address of the subscriber |
| **Returns:** | true or false depending on the result of erasing the subscription |

| **3. getSubscriptions ()** | |
|---|---|
| getSubscriptions method - returns all the existing subscriptions. | |
| **Parameters:** | |
| **Returns:** | an array with all the subscriptions |

| **4. clearSubscriptions (org.apache.axis.types.URI subscriber)** | |
|---|---|
| clearSubscriptions method - Erases all the subscriptions made by a subscriber with the given URI | |
| **Parameters:** | subscriber : address of the subscriber. |
| **Returns:** | true or false depending on the result of the operation |

| **5. publish (java.lang.String topicPattern, com.eu.hydra.eventmanager.axis.Part[] event)** | |
|---|---|
| publish method - publishes an event which will be received by everyone that is subscribed to the given topic. | |
| **Parameters:** | topicPattern: topic of the event.<br>event: event to be published. |
| **Returns:** | true or false depending on the result of the publishing |

| **6. notify (java.lang.String topic, com.eu.hydra.eventmanager.axis.subscriber.Part[] event)** |
|---|
| notify method - notifies a subscriber of the arrival of the new event. |

| | |
|---|---|
| **Parameters:** | topic: topic of the event published.<br>event: event published. |
| **Returns:** | true or false depending on the result of the notification. |

### 4.3.6 Application Network Manager

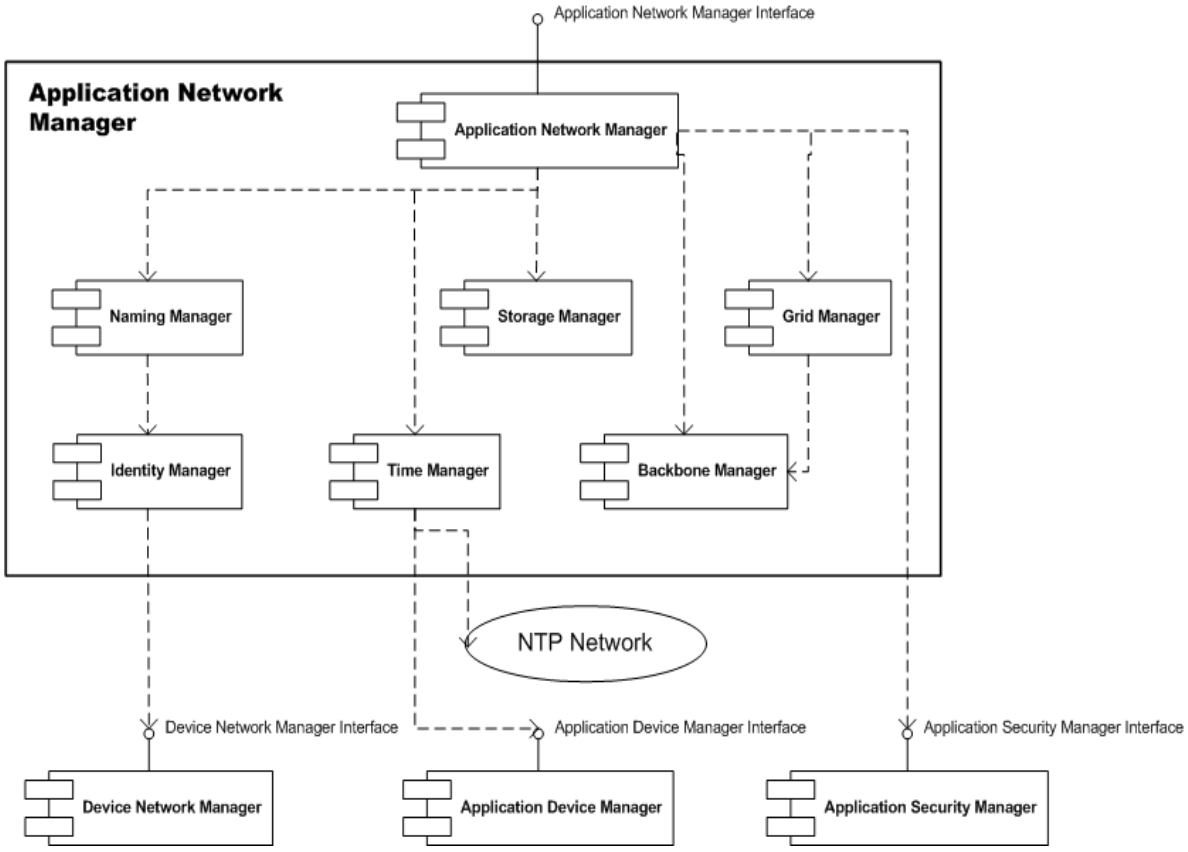


*Figure 16: Application Network Manager*

**Purpose:** The Application Network Manager is responsible for network management. It is in charge of providing a transparent view of the nodes in the application and to route the data to the appropriate node(s). It handles the HID of the nodes in the application. It also manages the names of the nodes for localisation purposes, GRID functionalities and a shared space for data storage. Additionally, it is in charge of synchronising the nodes in the network with referential time.

The subcomponents of this view are:

**Naming Manager:** responsible for managing information about the nodes and its position in the hierarchy of them (tree) that composes the node network architecture in an application. This information is used for location purposes.

**Storage Manager:** responsible for managing the application shared memory space.

**Grid Manager:** responsible for interfacing with the GRID This manager handles the communication with the grid service.

**Identity Manager:** responsible for managing the Hydra ID at application level.

**Time Manager:** responsible for synchronising the nodes of the application in the network with referential time

**Backbone Manager:** responsible for managing the information about the other services available in the network at application level.

**Main Functions:**

- Handle naming server
- Provide shared data space for nodes
- Provide Grid functionalities to locate resources
- Schedule node jobs
- 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:** Application Device Manager, Device Network Manager, Application Security Manager

## Public Functions

Public functions for the application view are same as the ones available in the device view. For function-descriptions please refer to the Device Network Manager section of this document.

### 4.3.7 Application Ontology Manager

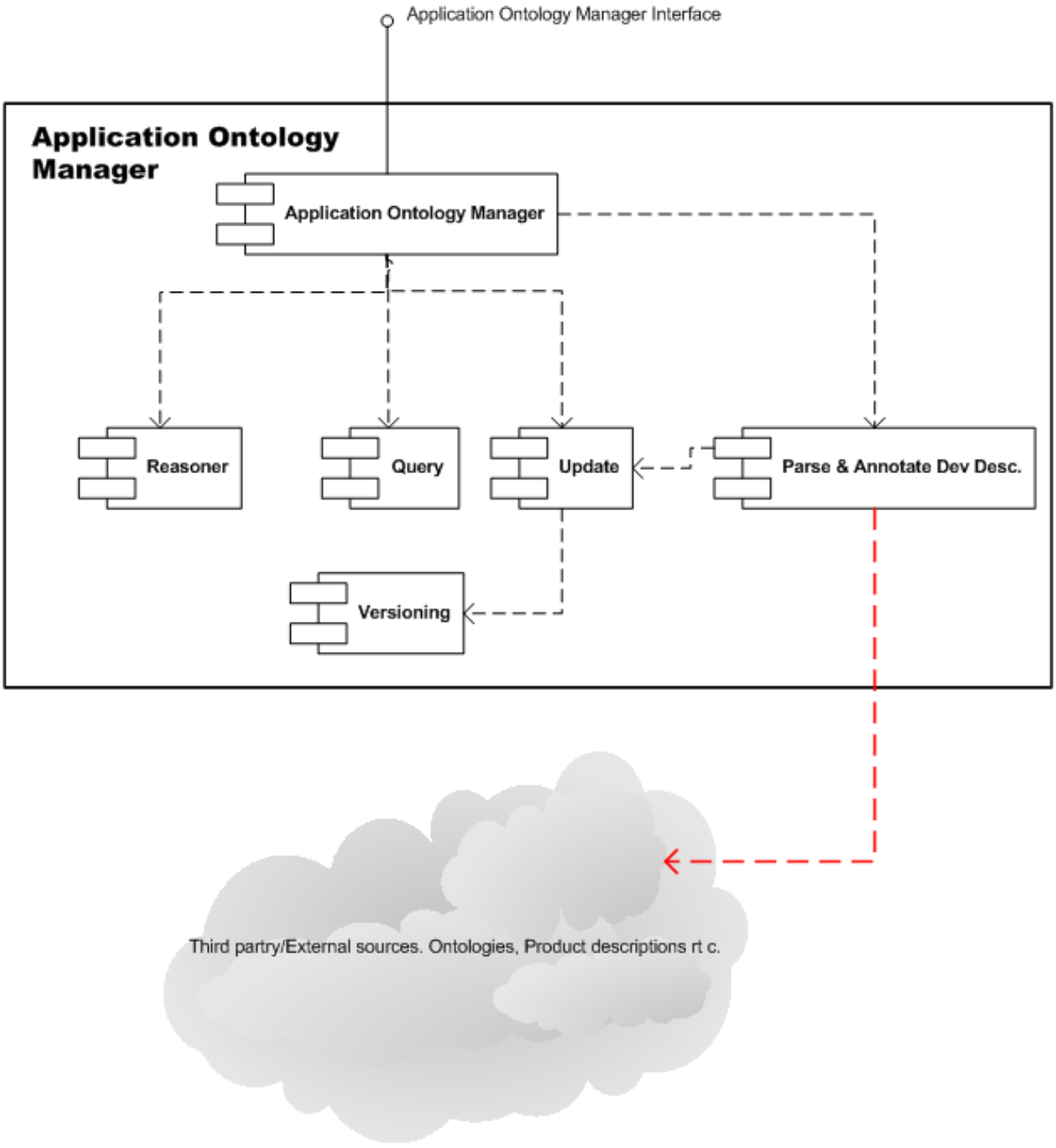


*Figure 17: Application Ontology Manager*

**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 and annotation
- Parsing and annotation of device description
- Search/Query function
- Update
- Ontology versioning
- Reasoner module

**Components:**

**Reasoner:** The reasoner module is responsible for reasoning about devices and their status and provides inferring 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 versions of the ontology. This includes different versions of devices and services.

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

**Dependencies:** External ontologies and product description databases

## Public Functions

| **1. getDeviceDescription (String deviceId)** | |
|---|---|
| Retrieves the device description from the ontology | |
| **Parameters:** | deviceId: The ontology device ID. |
| **Returns:** | An XML string containing the device description or the description of an error that occurred during method invocation |

| **2. getDeviceDescriptions()** | |
|---|---|
| Retrieves descriptions of all devices presented in the ontology | |
| **Parameters:** | |
| **Returns:** | An XML string containing the descriptions of all devices contained in the ontology or the description of error that occurred during method invocation. |

| **3. createNewDevice (String deviceType, String deviceId)** | |
|---|---|
| Creates the new device ontology instance | |
| **Parameters:** | deviceType: The device type specified as the name of the ontology concept in the device type hierarchy.<br>deviceId: The requested ontology device ID. |
| **Returns:** | An XML string containing the description of an unique and requested ontology ID of a newly created device |

| | |
|---|---|
| | instance or the description of an error that occurred during method invocation. |

| **4. setDeviceDescription (String deviceId, String description)** | |
|---|---|
| Sets or updates the basic device information in the ontology. | |
| **Parameters:** | deviceId: The ontology device ID<br>description: The XML string specifying the new basic information of the defined device. |
| **Returns:** | An XML string containing the update operation success or the description of an error, that occurred during method invocation |

| **5. getSupplierInfo (String deviceId)** | |
|---|---|
| Retrieves the device's supplier information from the ontology | |
| **Parameters:** | deviceId: The ontology device ID. |
| **Returns:** | An XML string containing the information of device supplier (manufacturer name and URL) or the description of an error, that occurred during method invocation. |

| **6. resolveError (String deviceType, String error)** | |
|---|---|
| Retrieves the errors for all devices of specified type | |
| **Parameters:** | deviceType: The device type specified as the name of the ontology concept in the device type hierarchy.<br>error: The human readable error description. |
| **Returns:** | An XML string containing the list of all devices, which contain the specified error or the description of an error that occurred during method invocation. For each device, the XML string contains the list of matching errors and for each error the related cause-remedy pairs. |

| **7. resolveErrorByCode (String deviceType, String error)** | |
|---|---|
| Retrieves the errors for all devices of specified type | |
| **Parameters:** | deviceType: The device type specified as the name of the ontology concept in the device type hierarchy.<br>error: The error specified by ontology error code. |

| | |
|---|---|
| | |
| **Returns:** | An XML string containing the list of all devices, which contain the specified error or the description of error that occurred during method invocation. For each device, the XML string contains the list of matching errors and for each error the related cause-remedy pairs. |

| **8. getDeviceError (String deviceId, String error)** | |
|---|---|
| Retrieves the errors for specified device | |
| **Parameters:** | deviceId: The ontology device ID.<br>error: The human readable error description. |
| **Returns:** | An XML string containing the list of all device errors matching the query or the description of error that occurred during method invocation. For each error, the XML string contains the list of related cause-remedy pairs. |

| **9. getDeviceErrorByCode (String deviceId, String error)** | |
|---|---|
| Retrieves the errors for specified device | |
| **Parameters:** | deviceId: The ontology device ID.<br>error: The error specified by ontology error code. |
| **Returns:** | An XML string containing the list of all device errors matching the query or the description of error that occurred during method invocation. For each error, the XML string contains the list of related cause-remedy pairs. |

| **10. setDeviceErrors (String deviceId, String errors)** | |
|---|---|
| Sets or updates the device errors in the ontology. | |
| **Parameters:** | deviceId: The ontology device ID.<br>description: The XML string specifying the device errors and cause-remedy pairs. |
| **Returns:** | An XML string containing the update operation success or the description of an error that occurred during method invocation. |

| 11. removeDeviceErrors (String deviceId) | |
|---|---|
| Removes the device errors from ontology | |
| **Parameters:** | deviceId: The ontology device ID. |
| **Returns:** | An XML string containing the remove operation success or the description of an error that occurred during method invocation. |

| 12. answer (String query) | |
|---|---|
| Utility method: retrieves the result of SPARQL (SPARQL Protocol and RDF Query Language) query. | |
| **Parameters:** | query: The SPARQL query. |
| **Returns:** | An XML string containing the result of executed SPARQL query or an error code. XML string and error codes are automatically generated by Jena [1]API. |

| 13. getDeviceTypes () | |
|---|---|
| Utility method: retrieves the names of device type hierarchy concepts. | |
| **Parameters:** | |
| **Returns:** | A string containing comma-separated list of names of device type hierarchy concepts |

[1] Jena is the Semantic Web Framework for Java: http://jena.sourceforge.net/

### 4.3.8 Application Policy Manager

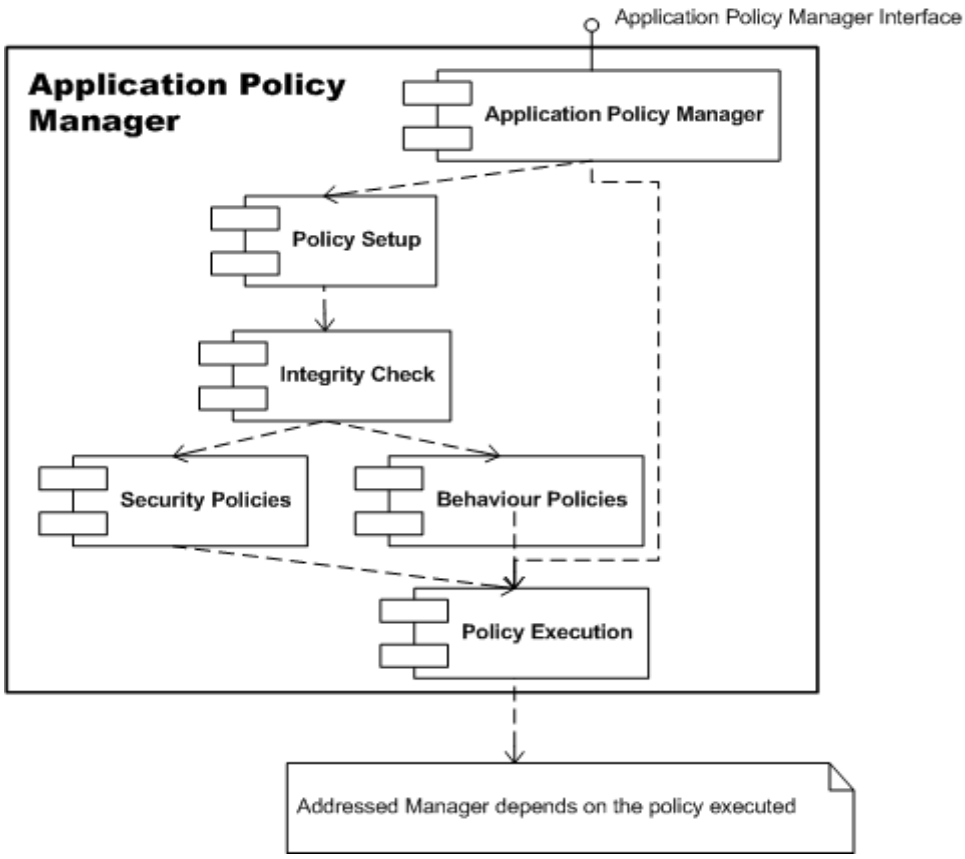


*Figure 18: Application Policy Manager*

**Purpose:** The Application Policy Manager is responsible for Policy Management and execution of the policies 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 ones 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.

## Public Functions

Other functions for the policy manager in the application view will be decided and implemented only after the second Hydra iteration.

| 1. | **getProtectionProfile (String contextInformation)** |
|---|---|

| Function to retrieve the protection profile, depending on the current content. | |
|---|---|
| **Parameters:** | contextInformation Current context information |
| **Returns:** | Protection Information requested (of data type Protection Information) |

### 4.3.9 Application Schedule Manager

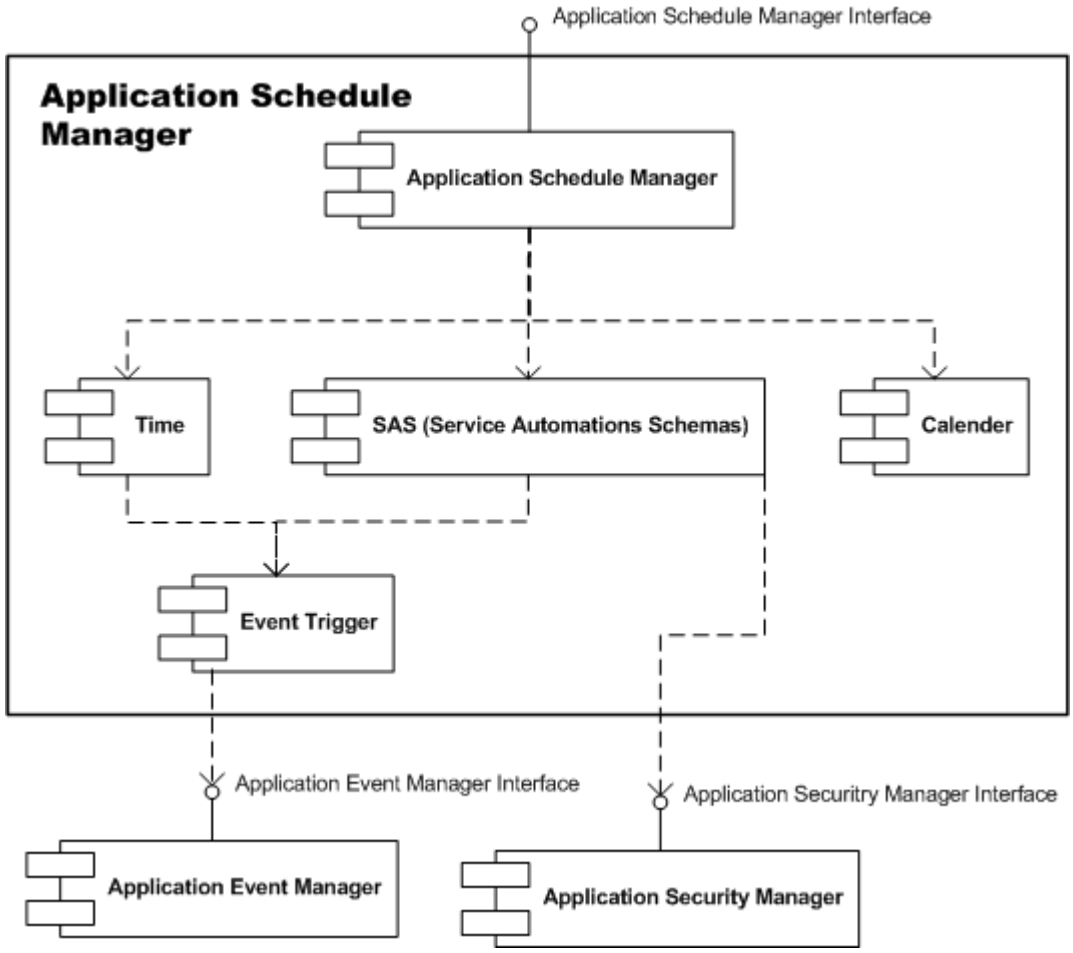


*Figure 19: Application Schedule Manager*

**Purpose:** The Application Schedule Manager implements service automation schemes (SAS) and performs event scheduling (including trigger functionality).

**Main Functions:**

- Event triggering (e.g., get cam image at noon weekdays)
- Calendar function
- Triggering
- Device coordination
- Schedule execution (based on service automation schemes)

**Dependencies:** Application Event Manager, Application Security Manager

## Public Functions

Public functions for schedule manager will be defined and implemented only after the 2nd Iteration.

### 4.3.10 Application Security Manager

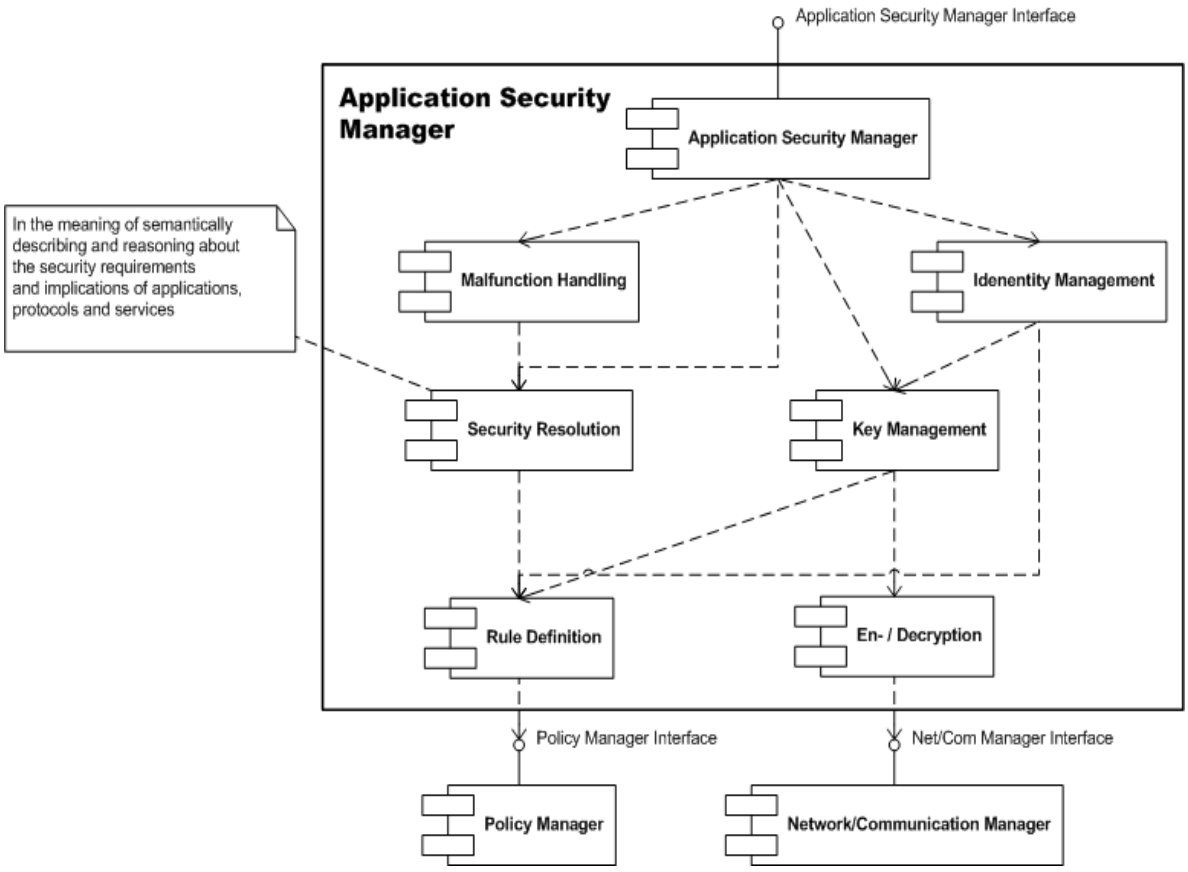


*Figure 20: Application Security Manager*

**Purpose**: The Application Security Manager realises 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 encryption and decryption of application and user specific data, regardless 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

## Public Functions

All Functions in Device view of security manager are available in the application view too. In Addition to that there are four more functions available in the application view, as given below. Not all of these methods are already fully implemented. In the second Hydra iteration, they might further be subject to some modifications, depending on the revision of the architecture.

| **1. createIdentity (String identity)** | |
|---|---|
| Function to create the Identity based on a value | |
| **Parameters:** | identity: ID value in string format (necessary for creating the identity) |
| **Returns:** | A unique identifier for the created identity or null in case of failure. |

| **2. deleteIdentity (java.lang.String identityID)** | |
|---|---|
| Function to delete the Identity | |
| **Parameters:** | identityID: Unique identifier of the identity |
| **Returns:** | String indicating the status of the execution of function |

| **3. retrieveCredentials (String storageID)** | |
|---|---|
| Function to retrieve the credentials based on the storage ID | |
| **Parameters:** | storageID: Storage ID in string format |
| **Returns:** | Credential value |

| **4. storeCredentials (CredentialType credentials)** | |
|---|---|
| Function to store generated credentials | |
| **Parameters:** | credentials: generated Credentials |
| **Returns:** | A unique storage ID of the stored credential as a string |

### 4.3.11 Application Service Manager

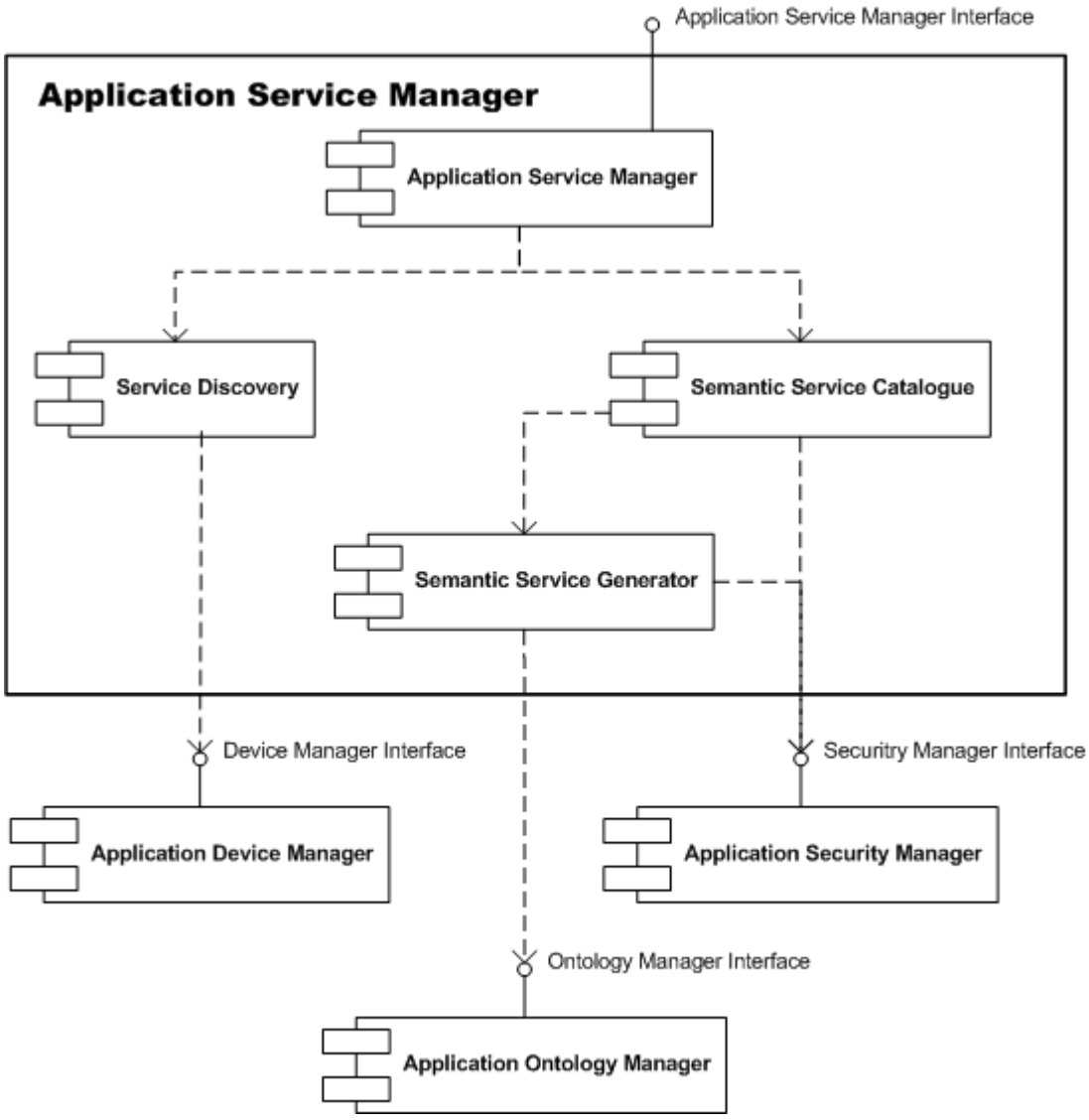


*Figure 21: Application Service Manager*

**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

## Public Functions

| **1. ProcessErrorMessage (XmlNode theMessage)** | |
|---|---|
| Processes an error message | |
| **Parameters:** | theMessage: The error message as an XML Node. |
| **Returns:** | A description of the error |

| **2. ProcessErrorMessageString (string theMessage)** | |
|---|---|
| Processes an error message | |
| **Parameters:** | theMessage: The error message as a string. |
| **Returns:** | A description of the error |

| **3. HasService (string service)** | |
|---|---|
| Checks if a service is available | |
| **Parameters:** | service: The service name. |
| **Returns:** | True if service is available otherwise false |

| **4. GetServiceDescription (string devicetype, string serviceid)** | |
|---|---|
| Retrieves a device description | |
| **Parameters:** | devicetype: The device type as a string.<br>serviceid: The service ID as a string. |
| **Returns:** | A string containing a service description in XML format |

| **5. GetServiceDescriptionAsXML (string devicetype, string serviceid)** |
|---|

| Retrieves a device description | |
|---|---|
| **Parameters:** | devicetype: The device type as a string.<br>serviceid: The service ID as a string. |
| **Returns:** | An XML Node containing a service description |

| **6. GetDevices (string type)** | |
|---|---|
| Retrieves a list of available devices | |
| **Parameters:** | type: The device type as a string. |
| **Returns:** | A string containing the list of available devices in XML format |

| **7. GetDevicesAsXML (string type)** | |
|---|---|
| Retrieves a list of available devices | |
| **Parameters:** | type: The device type as a string. |
| **Returns:** | An XML Node containing the list of available devices |

| **8. SendServiceRequest (string serviceRequest)** | |
|---|---|
| Function to send a service request to be passed to the Context Manager. | |
| **Parameters:** | serviceRequest: The service request as a string. |
| **Returns:** | A string containing the service request |

| **9. SendErrorConfirmationAndCredentials (string errorConfirmation, string credentials)** | |
|---|---|
| Function to send an error confirmation with credentials. It passes the request to the Context Manager. | |
| **Parameters:** | errorConfirmation: The error confirmation as a string.<br>credentials: The credentials as a string. |
| **Returns:** | A string containing the error confirmation request |

| **10. SendServiceOrderAndCredentials (string serviceOrder, string credentials, string deviceid)** |
|---|
| Function to send a service order with credentials for a device. It passes this |

| request to the Context Manager. | |
|---|---|
| **Parameters:** | serviceOrder: The service order as a string.<br>credentials: The credentials as a string.<br>deviceid: The device id as a string. |
| **Returns:** | A string containing the service order |

| **11. InitiateErrorAnalysis (string errorAnalysis, string deviceid)** | |
|---|---|
| Function to initiate error analysis for a device. Passes the request to the Context Manager. | |
| **Parameters:** | errorAnalysis: The error as a string.<br>deviceid: The device ID as a string |
| **Returns:** | A string containing the error analysis request |

| **12. SendErrorReport (string errorReport, string provider, string deviceid)** | |
|---|---|
| Function to send an error report for a device. It passes this request to the Context Manager | |
| **Parameters:** | errorReport: The error report as a string.<br>provider: The provider as a string.<br>deviceid: The device id as a string. |
| **Returns:** | A string containing the error report request |

| **13. SendBill (string bill, string serviceagent)** | |
|---|---|
| Function to send a bill for a service agent. It passes this request to the Context Manager | |
| **Parameters:** | bill: The bill as a string.<br>serviceagent: The service agent ID as a string. |
| **Returns:** | A string containing the send bill request |

| **14. SendGetStatus (string deviceid)** | |
|---|---|
| Sends a get status request for a device. Passes the request to the Context Manager | |
| **Parameters:** | Deviceid: The device ID as a string. |
| **Returns:** | A string containing the current status of the device |

| **15. ReceiveFirmWare (string deviceID, byte[] firmware)** | |
|---|---|
| Receives new firmware for a device | |
| **Parameters:** | deviceID: The device ID as a string.<br>firmware: The new firmware as a byte array. |
| **Returns:** | A boolean indicating if the new firmware was properly received |

| **16. ResolveErrorWithDeviceID (string deviceID, string errorCode, string errorStr)** | |
|---|---|
| Resolves an error for a device | |
| **Parameters:** | deviceID: The device ID as a string.<br>errorCode: The error code as a string.<br>errorStr: The error as string |
| **Returns:** | A string containing the resolved error information |

| **17. ResolveErrorWithDeviceIDAsXml (string deviceID, string errorCode, string errorStr)** | |
|---|---|
| Resolves an error for a device | |
| **Parameters:** | deviceID: The device ID as a string.<br>errorCode: The error code as a string.<br>errorStr: The error as string |
| **Returns:** | An XML Document containing the resolved error information |

### 4.3.12 Application Session Manager

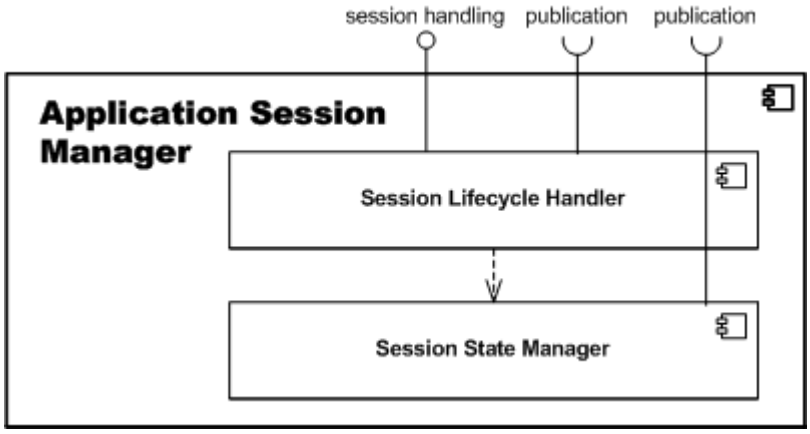


*Figure 22: Application Session Manager*

**Purpose:** The Application Session Manager is responsible for maintaining sessions and session-specific state.

Sessions are used to identify client-specific "applications" on a device in terms of, e.g., the data that is needed for a particular interaction. Moreover, the session manager is responsible for creating, changing, and deleting sessions and while doing so notifying interested listeners (through the Application Event Manager).

**Main Functions:**

- Session setup and lifecycle management
- Notifications on session state change

**Dependencies:** Application Event Manager

## Public Functions

Session management is now a function of the network manager. So please refer to the Device network Manager for function descriptions in this document.

# 5. Summary

This primary goal of this training document is to provide an overview of the Hydra architecture to the external developers. It elucidates the architecture and functions available for each manager, thereby providing the developers and manufacturers, a glimpse of what can be achieved through Hydra. However, it must be noted that this deliverable has been submitted in month 18 and so it is not the complete Hydra-training document, but a first step in providing hydra-training to external developers.

Hydra is an ongoing project and innovative ideas are introduced at every stage of this project based on the feedback from developers. We are planning to release the SDK as an eclipse plugin so as to provide an 'easy-to-use' framework to the developers. By choosing a widely used IDE, we are trying to avoid the huge learning curve needed for the developers to get used to a new IDE. In the next version of training document, the function descriptions will be grouped together to form the API-Help document and the main training document will focus on providing solutions using these functions, elucidating sample implementations.

# 6. Bibliography

[1] Hydra Website, www.hydra.eu.com

[2] Hydra Deliverable, *D2.1 Scenarios for usage of Hydra in 3 different Domains,* 2006

[3] Hydra Deliverable, *D3.3 Draft of architectural specification,* 2007

[4] Hydra Deliverable, *D12.1 Internal technology workshops teaching material*, 2006

[5] Hydra Deliverable, *D5.3 Draft of Wireless Network Implementation Description*, , 2007

[6] Hydra Deliverable, *D5.4 Draft of Wireless Device Integration Description*, 2007