FlexiblePower Application Infrastructure Detailed Functional Design

Version 1.0
February 2013

Table of Contents

Table of Figures

1 Introduction

1.1 Scope

This document contains the detailed functional design (version 0.7) of the FLEXIBLEPOWER APPLICATION INFRASTRUCTURE.

The design describes the actor and system entities of the FLEXIBLEPOWER APPLICATION INFRASTRUCTURE platform, with their responsibilities, interactions, communications and internal organization. The platform is realized by the FLEXIBLEPOWER APPLICATION INFRASTRUCTURE, which includes an application store, management center and home boxes with a runtime software framework.

The top-level functional decomposition of the infrastructure is presented. This translates into technical systems and components. The deeper decomposition presented in this design is of a functional nature only, it does not necessarily imply a similar technical decomposition. The chosen technical architecture and design options are described in a separate technical design document.

1.2 Notations

In this design we introduce many actors, system entities and functional concepts, for which we use the following notations.

- *Full names* have multiple terms in camel case; used when first introducing the name in text.
  - An example is "FLEXIBLEPOWER MANAGEMENT CENTER".
- *Abbreviation style names* are used for brevity and consist of FP term where term is an abbreviation or camel case.
  - An example is "FP MANAGEMENTCENTER".
- When clear from the context we use single words or abbreviations as *shorthand style*.
  - An example is "management center" or even "MC".

1.3 Concept outline

1.3.1 FlexiblePower Application Infrastructure

The FLEXIBLEPOWER APPLICATION INFRASTRUCTURE is an open platform focused on the end-users, which can be households as well as companies. For the energy management a home box (FLEXIBLEPOWER HOME BOX) resides at the end-user premise(s) to monitor and control its energy consumption and generation devices and appliances. It can interact with parties from the energy market and distribution. Third parties can develop energy management applications that can be deployed on the box; applications are offered in an application store (FLEXIBLEPOWER APPSTORE). A central management center (FLEXIBLEPOWER MANAGEMENT CENTER) provides remote device[1] and service management. This concept is illustrated in the next figure.

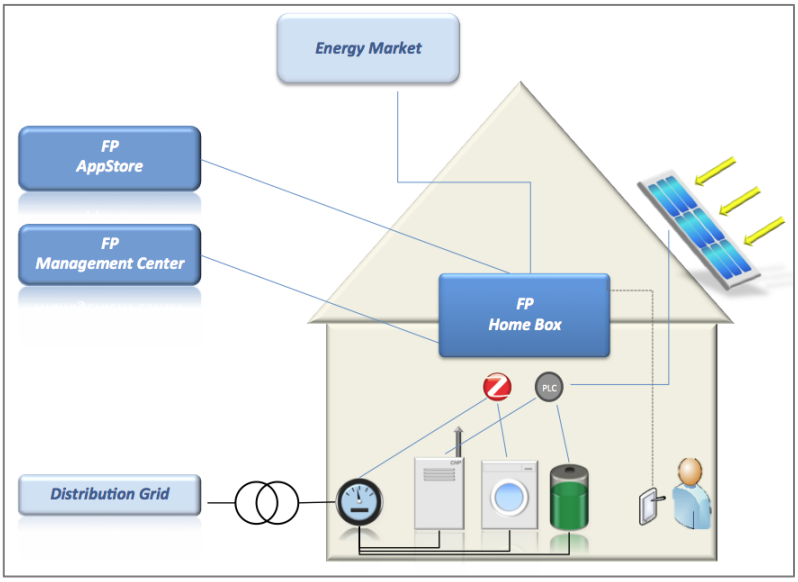


Figure 1-1 FLEXIBLEPOWER APPLICATION INFRASTRUCTURE concept

1.3.2 FlexiblePower Application Infrastructure software on the FlexiblePower Home Box

The FLEXIBLEPOWER HOMEBOX (FP HOMEBOX) has FLEXIBLEPOWER APPLICATION INFRASTRUCTURE software installed, which can be extended by installing additional applications from the store. This software environment is called the FLEXIBLEPOWER RUNTIME (FP RUNTIME).

The term FP HOMEBOX often refers to the hardware on which the FP RUNTIME software is installed. Since the design also allows for virtual (non-hardware) FP HOMEBOXES, we mostly use the term FLEXIBLEPOWER NODE or FP NODE (or briefly "node") to refer to the home box from a functional or software perspective.

The node has interfaces to its ecosystem:

- An application store interface (FP AS interface) to communicate with the FP APPSTORE
- A service and management interface (FP SM interface) to communicate with the FP MANAGEMENTCENTER

[1] Note that "device" here refers to the FP HomeBox devices, not to the home energy devices and appliances.

- Interfaces (FP UI interface) to support interactions by Users with the Home Box to manage App and User Profiles/Preferences, using a GUI[2] on an end-user device.[3]

The node has provisions for applications to perform energy management:

- FP CONNECT, which support the connection between home appliances/devices and the home box via protocols such as ZigBee or PLC.
- FP RAL (FLEXIBLEPOWER RESOURCE ABSTRACTION LAYER), which is a resource abstraction layer to represent and manage physical devices and appliances as abstract resources, which expose their energy flexibility information. The RAL determines the flexibility of the resources and combines them with user preferences.
- FP RAI (FLEXIBLEPOWER RUNTIME APPLICATION INTERFACE), which specifies a uniform way to expose energy flexibility to applications.
- Provisions that allow installation and operation of energy management applications (Energy Apps), which use the FP RAI information to perform energy management and interact with the energy market.[4]

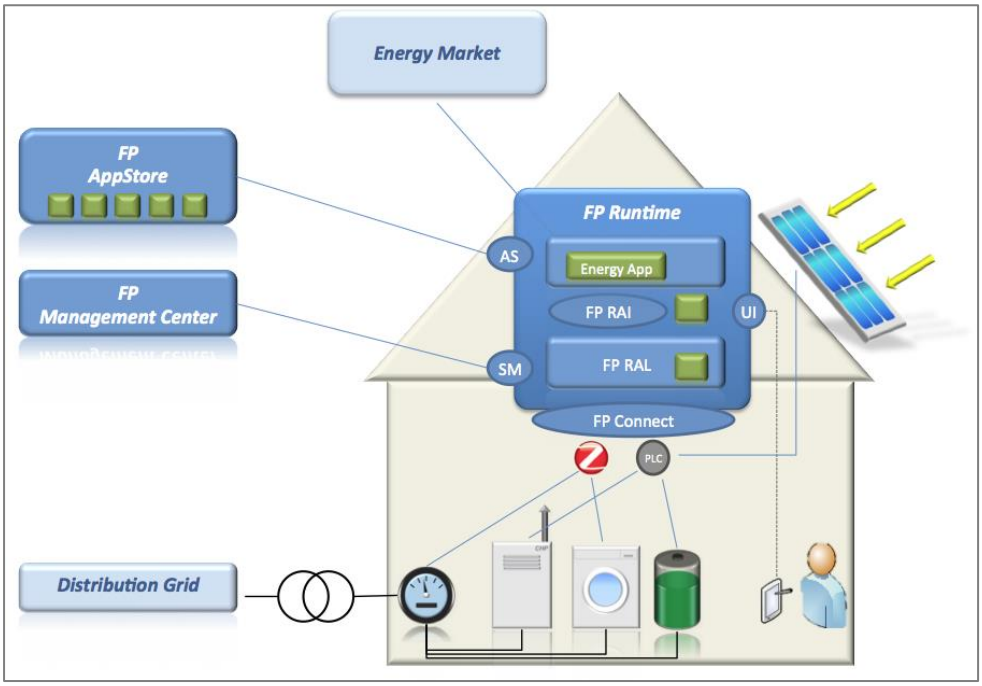


Figure 1-2 Concepts of FP Application Infrastructure software on the FP HomeBox

[2] Graphical User Interface
[3] User devices can be PC's, smartphones, tablets, …
[4] Note that these provisions will be referred to later in this design document by the term "FlexiblePower Energy Application Layer" or "FP EAL". Energy applications are installed in the energy application layer. Note also that there are other kinds of apps that reside elsewhere; for example resource-related applications, such as Resource Manager Apps or Resource Driver Apps, are applications that reside in the RAL. Figure 1.2 shows these other apps as additional green boxes (either in the RAL or elsewhere in the FP Runtime) besides the energy app (in the EAL).

1.3.3 *PowerMatcher as energy application on the FlexiblePower Application Infrastructure.*

The FP Application Infrastructure is an open framework that can serve different types of energy management applications. The specific PowerMatcher technology for energy management can be deployed on the FP Application Infrastructure. As shown in the next figure, it consists of an energy application on the FP HomeBox, and a PowerMatcher Auctioneer deployed in the energy market.

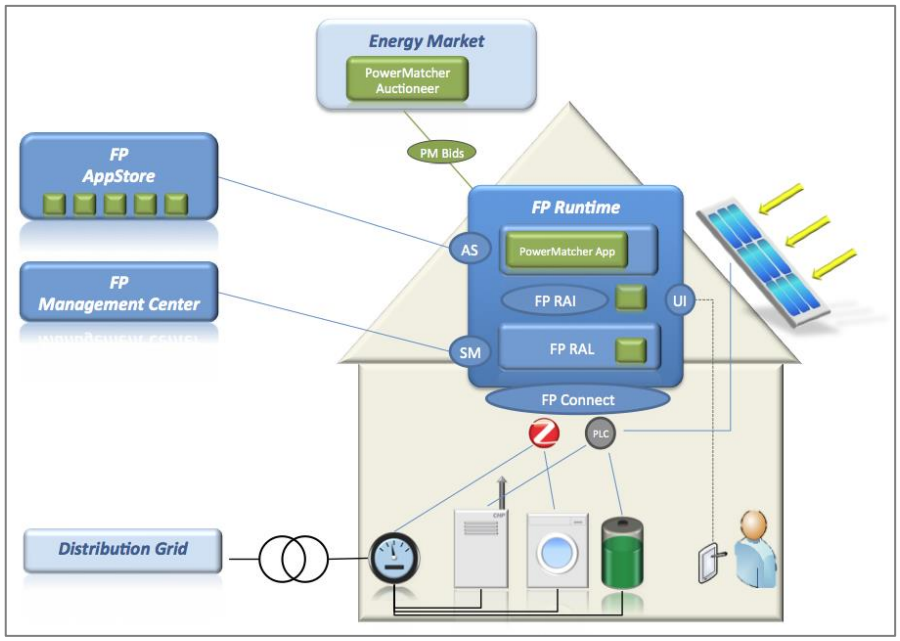


Figure 1-3 PowerMatcher on the FP Application Infrastructure

The Concentrator Agent and Device Agents are components of the PowerMatcher App; the Auctioneer Agent is part of the PowerMatcher Auctioneer. This is illustrated in the figure below; the PM Bids agent communication is shown as Bid/Price messages.

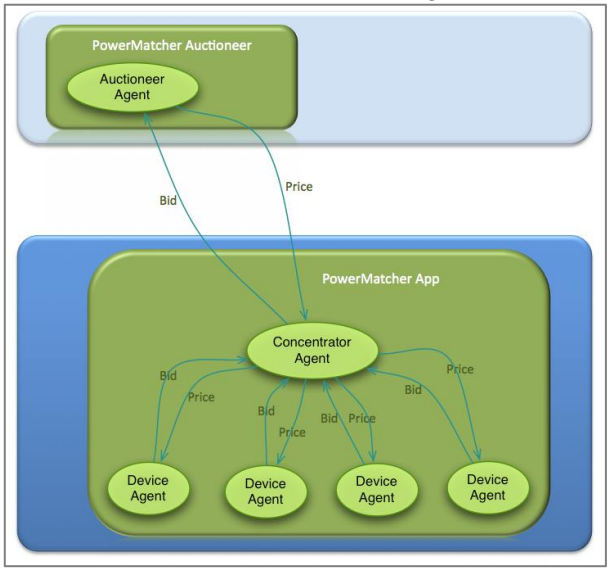


Figure 1-4 PowerMatcher Agents

1.3.4 *Resource Abstraction Layer*

Within the vision of an emerging 'end-user driven' competitive energy services economy, fading boundaries between energy users and producers, appliances and services heterogeneity and paramount freedom of choice for end-users, the FLEXIBLEPOWER APPLICATION INFRASTRUCTURE supports the mission to prevent and limit negative consequences of "format-wars" on interfaces for appliances and energy services, create interoperability between heterogeneous appliances and energy services, create a low-cost and level playing field for energy service providers and support a broad and quick adoption of interoperable energy services and appliances. The FLEXIBLEPOWER APPLICATION INFRASTRUCTURE aligns with the strategy to define open and generic interfaces, promote adoption by developing industry-enabling tools, to create a large partner-network and to stimulate end-user interest and trust.

FLEXIBLEPOWER APPLICATION INFRASTRUCTURE is an effort to address the challenge of creating interoperability in a heterogeneous context of different smart grid services and market approaches exploiting flexibility, different types of appliances capable of delivering flexibility and many different protocols.

It therefore has as a main focus on the abstraction of types of devices into resources, by first abstracting appliance and manufacturer specifics into "resource drivers", secondly to abstract further into "resource managers" that model energetic flexibility of a device and expose it in a uniform way. Analogous to a HAL (Hardware Abstraction Layer) that sits between hardware and software on computer, the FLEXIBLEPOWER APPLICATION INFRASTRUCTURE aims at realizing a **Resource Abstraction Layer** (RAL) that sits between applications and resources, providing abstraction of capabilities of resources and managing access of applications to resources. End-user applications combine the energy flexibility information offered by the abstraction layer with user preferences and settings.

1.3.5 *Abstraction of Devices into Resources for Applications*

With the previous considerations in mind, the next diagram shows on a high-level where the main abstraction is made. The application level is called "ENERGY APPLICATION"; the abstraction of a device into a Resource Manager and a Resource Driver is called a "Resource". The Energy Application mainly reasons with or "coordinates" energetic flexibility information of resources (exposed as FP RAI information by its resource manager) and offerings of an energy market participant, in combination with overall consumption/generation information, user energy target settings, sensor information and other third party information.

The main abstraction boundary is situated between the energy application(s) and the resources, represented in the figure as middle (green) boundary. It hides protocol and device specifics and exposes energy flexibility information to applications. This abstraction is realized by the FP RAI and FP RAL.

The energy application interacts with a specific market participant using its protocol to exchange information. The interface is not abstracted; therefore it is shown as "specific" top boundary (blue).[5]

The "specific" boundary (bottom blue) represents the specifics of interfaces and protocols to work with a device. These are not visible for the applications but abstracted by the resource driver of a resource.

The FlexiblePower Application Infrastructure has its main focus on offering "**resource abstraction" for applications**, which translates in an abstraction of devices into resources and the provision of a generic and uniform way for an application to reason with their energy flexibility information and in their own specific way with an energy market participant.

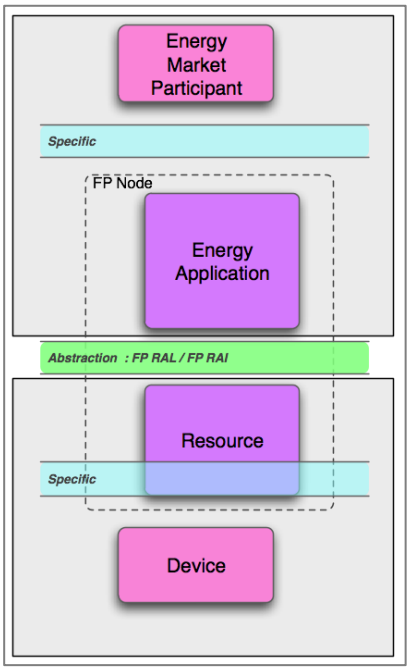


Figure 1-5 Abstraction of devices from applications

Side note:

FlexiblePower Application Infrastructure has not the focus on also offering "market abstraction" for applications.

An abstraction of that kind would also shield the energy market participant specifics from the application and result in a model with two main abstraction boundaries, as shown in the figure on the right.

For applications to have an abstract view on market participants, one would need to introduce a second abstraction boundary, so that the application not only works with an abstract resource exposing uniformly its energy flexibility information, but now also interacts with an abstract market participant, exposing uniformly its information such as offerings, proposals, etc.

This would be an additional focus and realize a MARKET ABSTRACTION LAYER[6] (MAL) for applications.

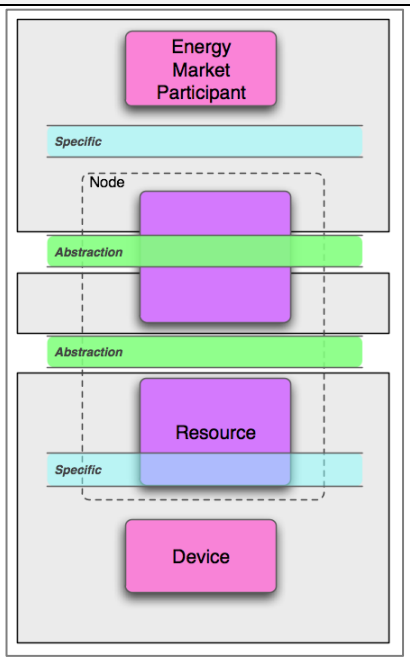


Figure 1-6 Alternative concept with more abstractions

[5] Note that in the case of installing the PowerMatcher App, the specific protocol used is called PM Bids, where Bid and Price information is exchanged between the PowerMatcher App and the PowerMatcher Auctioneer.

[6] This design does not provide a market abstraction layer, in compliance with the requirements from stakeholders, as described in the FLEXIBLEPOWER APPLICATION INFRASTRUCTURE Requirements Specification. The side note(s) are only included here to explain why global design choices on abstraction boundaries were made.

1.3.6 *Decision Layers in FLEXIBLEPOWER APPLICATION INFRASTRUCTURE*

The focus on “resource abstraction for applications” results in two decision layers in the FP Node software (the FP Runtime) on an FP HomeBox.

The next figure illustrates how the functional design, contained in this document, translates the high-level principles of resource abstraction as introduced in the previous paragraph.

- The **Resource Abstraction Layer** (FP RAL) of a node contains the Resources, which exist of a Resource Manager and a Resource Driver.
- The **Energy Application Layer** (FP EAL) contains the Energy Applications that reason with the exposed energy flexibilities (expressed by FP RAI) and (an) Energy Market Participant(s).

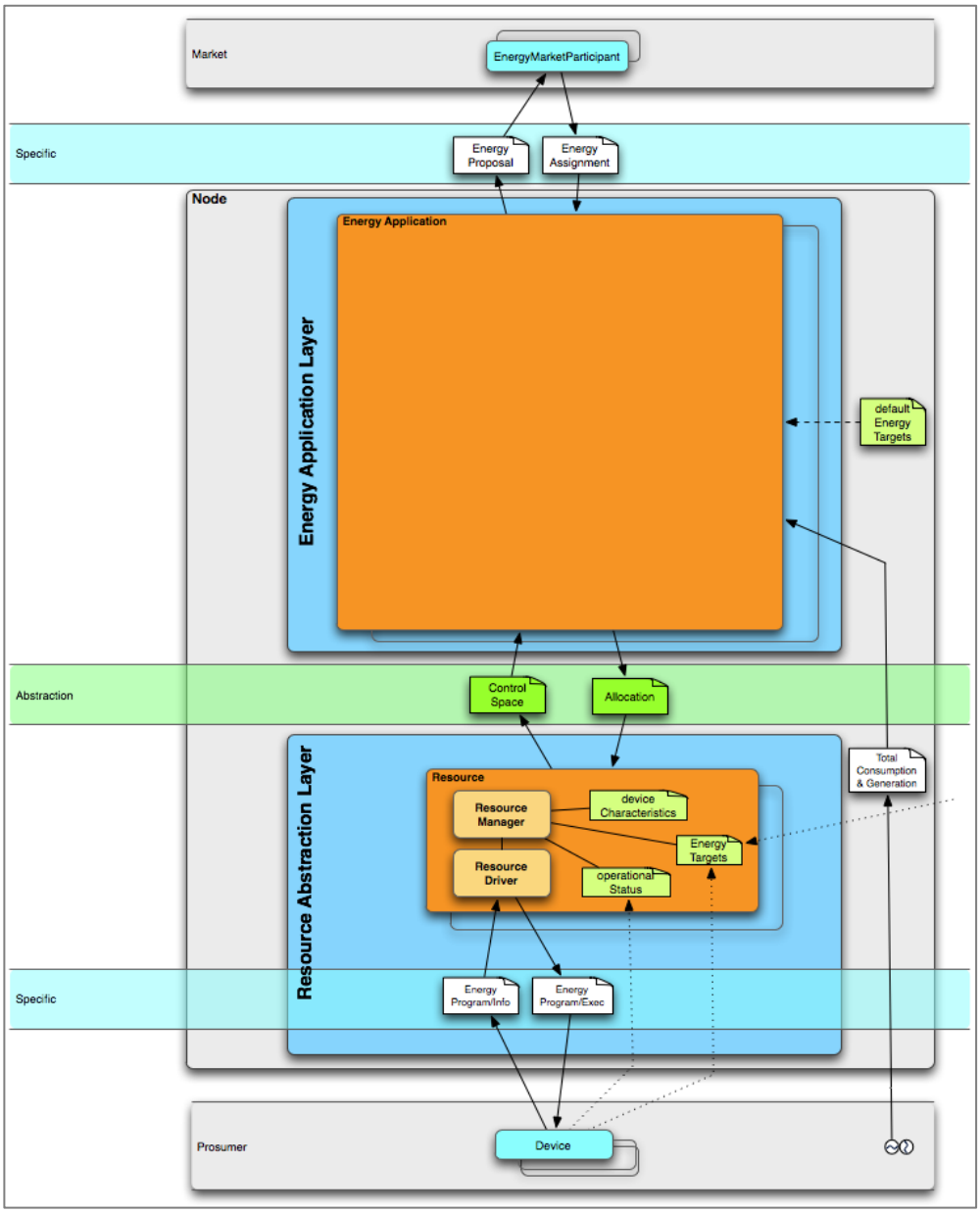


Figure 1-7 View on node with 2 decision layers, 1 abstraction boundary

Side note:

Designing for an additional “market abstraction for applications” focus, one would need three layers, since two abstraction boundaries (resource and market) would result in a three- layered model.

The **Resource Abstraction Layer** would be present as before, but the **Energy Application Layer** would now be split into an **Organizer Layer** (containing the application) and a **Participator layer**. This layer would contain abstract representations of energy market participants (Participators), similarly as for resources containing a manager and a driver to realize the abstraction.

This approach is shown in the figure on the right.

We included this side note and figure for clarity. It is however **not the focus** of the FP Application Infrastructure and therefore not included in the design.

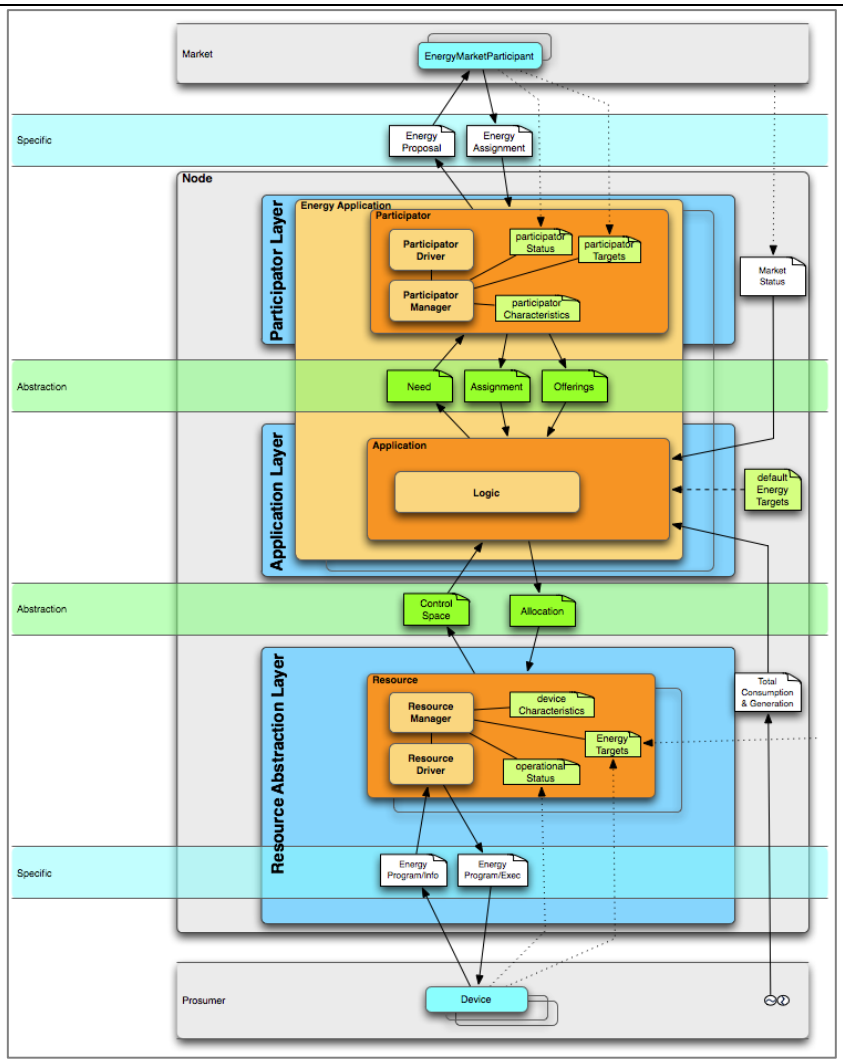


Figure 1-8 View on node in alternative concept.

Further details are discussed in this detailed functional design document.

Note that where we situated the main “application” as ENERGY APPLICATION in this introduction, this is only one of many examples where a so-called “Apps” can be deployed in the FP NODE software. The design will formally introduce Apps, their categories and their deployment, how the FP APPSTORE offers them, etc. This concept outline only serves as an introduction.

2 FlexiblePower Application Infrastructure Functional Concepts

2.1 Notations and conventions

The following are concepts as defined in the FLEXIBLEPOWER APPLICATION INFRASTRUCTURE Requirements and Vision documents, and of which some were introduced in the concept outline of the previous chapter. In the functional architecture we translate these concepts into entities and artifacts to specify the design in much more detail. This chapter serves as a summary of the concepts introduced, to provide a basis for the detailed functional design in the next chapters. Concepts are indicated with *Italics*, entities will be indicated in SMALLCAPS.

2.2 Basic Concepts

2.2.1 Overview

Concept	Description
Producer	Entity, independent or public, which generates electricity from other forms of energy.
Large Power Producer	Large scale power producer
DG Operator	Distributed Generation Operator is a *Producer* with small-scale generation, also called an operator of a Distributed Energy Resource (DER) system.
TSO	The Transmission System Operator (TSO) operates the *Transmission Grid* and its interconnections in a given area; it ensures the long-term stability of the system to meet reasonable demands for the transmission of electricity.
Transmission System	The *Transmission System* refers to the entire system, operated by the TSO, including the network, which is called the *Transmission Grid*.
Transmission Grid	The term "transmission" refers to the transport of electricity on the high-voltage interconnected system, the *Transmission Grid*.

A *TSO* is responsible for maintenance and development of the *Transmission Grid*, but also provides system services[7]. Electricity transmission and distribution is considered as "copper-plate".

Concept	Description
DSO	The Distribution System Operator (DSO) operates the *Distribution Grid* in a given area and its connections to the *Transmission Grid*, it ensures the long-term stability of the system to meet reasonable demands for the distribution of electricity. The DSO distributes the electricity via the Distribution Grid to the final customers. The consumer (prosumer) pays a "connection charge" and a "use of system charge" to the DSO for the delivery and the system services.
Distribution System	The *Distribution System* refers to the entire system, operated by the *DSO*, including the network, which is called the *Distribution Grid*.
Distribution Grid	A *Distribution Grid* or distribution system's network carries electricity from the *Transmission Grid* and delivers it to customers. The term "distribution" refers to the transport of electricity on the high-voltage, medium-voltage and low voltage distribution systems, with a view to its delivery to customers, not including supply.
Prosumer	Households or industrial facilities that purchase and consume electricity are called *prosumers*. They can also have on-site production of DG electricity; therefore they are called *prosumers* instead of consumers.
Supplier	Responsible for the sale of electricity to customers (not always same entity as the producer)

[7] "Balancing services" (e.g. compensating differences in demand/supply), "reserve capacity" (e.g. compensating shortfall in power generating capacity), "power quality" (e.g. frequency control), "reactive power supply" and "black start capability".

Wholesale Market	Market of electricity as commodity, where it is traded between different actors.
Third Party Information Provider	A *Third Party Information Provider* is a party that provides useful information for energy management of a Prosumer. For example a weather service can supply weather forecasts that can be taken into account for energy supply/demand forecasting of a Prosumer.[8]

The FLEXIBLEPOWER APPLICATION INFRASTRUCTURE does not need to model all concepts and interactions of the physical and commodity system, it only models the concepts relevant for its operation.

2.2.2 *Notes on the physical system*

Large Power Producers generate electricity that is fed into the Transition Grid. By the (regulated) agreement between the Large Power Producer and the TSO, the power producer pays a "connection charge" (sometimes also a "use of system charge", for the transport of the produced electricity via the Transmission Grid by the TSO. The TSO transports the produced electricity to the DSO's via the Transition Grid. The DSO distributes the electricity via the Distribution System to the final consumers. The prosumer pays a "connection charge" and a "use of system charge" to the DSO for the delivery of the electricity and system services. Electricity generated by DG Operators is directly fed into the Distribution System. For this there is an agreement between DG Operator and the DSO. The DG Operator pays a "connection charge" and sometimes a "use of system charge" to the DSO for the electricity transport and system services. Most of the electricity from DG Operators is distributed by the DSO to consumers. When supply exceeds demand the surplus of electricity is fed upwards into the transmission grid, after which the TSO transports it to other distribution networks. In case of so-called "autoproduction" of DG electricity, the electricity produced on-site by a prosumer is directly consumed by the prosumer, omitting the commodity purchase and sales process through the energy supplier.

2.2.3 *Notes on the commodity system*

Large Power Producers and some large DG Operators offer the commodity on the wholesale market, where it is traded between different actors. The trade provides a payment for the produced electricity. Large electricity consumers (e.g. industrial customers) can buy commodity directly on the wholesale market. Smaller electricity consumers (e.g. households) buy commodity from a supplier, which buys the commodity on the wholesale market. The supplier, who is responsible for the sale of electricity to customers, is not always the same entity as the producer. A supplier can also be a wholesale customer or independent trader who purchases electricity with the purpose to resell it within the system. A supplier can therefore, instead of selling to a final consumer, sell to another customer in the wholesale system, which thereby acts as supplier to smaller customers. A supplier can also extract the commodity directly via (small) DG Operators and subsequently deliver the commodity from the wholesale market and the DG Operators to the consumers, who pay for it. Energy suppliers often have contracted more that they plan to offer to consumers; therefore there is a commodity stream backwards to the wholesale market.

2.3 Platform Concepts

2.3.1 *System Concepts*

Concept	Description
FP Application Infrastructure	The FlexiblePower Application Infrastructure (FP Application Infrastructure) is the term used to identify the entire FlexiblePower Application Infrastructure platform; including its hardware FP HomeBoxes, their FP Node software (also called FP

[8] Third Party Information Providers will gain access to the Prosumer to provide their information via Apps installed at the FP HomeBox of the Prosumer. A weather service provider can offer a weather app, which a Prosumer can install on its FP HomeBox to receive weather forecast information for use in its energy management.

	Runtime), the management center and application store, and all of the software, specifications, protocols, operational models and guidelines.
FP HomeBox, FP Runtime, FP Node	An FP HomeBox is a system that is deployed at the location of the *Prosumer* and controls/monitors energy consumption and generation devices. It coordinates the generation & consumption, it communicates with parties from the *Wholesale Market*, the *Supplier*(s), the *DSO* and *Third Party Information Provider*s. A Prosumer can have multiple locations, therefore multiple nodes. FP HomeBox refers to the hardware on which the FlexiblePower Application Infrastructure software is installed. The term FP Runtime is used to identify installed software, consisting of the FP Application Infrastructure software and additional energy applications (see further). The design allows for virtual (non-hardware) FP HomeBoxes, therefore we mostly use the term FP Node or briefly "node" to refer to the home box from a functional perspective.
FP ManagementCenter	The FP ManagementCenter remotely manages FP HomeBoxes and its FP Node software. The operator can be for example the DSO, although this is not required.
FP AppStore	Third parties can develop modules for the FlexiblePower Application Infrastructure and offer these in the FP AppStore, where *Prosumers* can buy and download them for installation on their *FP HomeBox*.

2.3.2 *Node Concepts*

Concept	Description
Resource Manager	A physical consumption/generation device of the *Prosumer* that is monitored/controlled by the *FP Node* is abstractly represented as a *Resource Manager*, responsible for proposing so called "Control Spaces" towards the energy application(s). With this, the Resource Manager provides the energetic possibilities of the connected device.
Resource Driver	A *Resource Driver* is responsible for actual control of connected devices and communication with a *Resource Manager*. As most devices will have proprietary interfaces, the *Resource Drivers* will very specific.
Energy Application	An *Energy Application* is a decision logic component of an FP Node that aggregates the energy patterns of all individually connected resources; it can negotiate between the node and a participant of the Wholesale Market; it performs coordination between the energy consumption/generation of resources and their flexibility, the energy targets defined by the end-user and the offerings from participants of the market. This coordination results in allocations to control the resources. Note that an "energy application" is only one particular kind of application type one can install; there are many other different types as well.
Security Management Framework	An FP HomeBox (its *FP Runtime software)* has a framework in place to ensure that it works and communicates securely and that it has local management in place to operate highly independent from the *FP ManagementCenter*. The combined security and management framework is called the *Security Management Framework.*
Graphical User Interface	End users (members of a Prosumer) can interact with the FP Node with a GUI to provide energy targets/settings and to get feedback on the energy consumption/generation and optimization performed.

2.3.3 *Data Concepts*

Concept	Description
Energy Proposal	An *Energy Proposal* is an expression of energy consumption/generation needs and its flexibility, which the (energy application of the node) offers to a specific *Wholesale Market* participant.
Energy Assignment	An *Energy Assignment* is a response from a *Wholesale Market* participant on an *Energy Proposal*.

Control Space	A Control Space is a generic representation of the energetic possibilities of a device, produced by its Resource Manager, collected by the energy application.
Allocation	An Allocation is a generic representation of an energetic allocation pattern, derived by the energy application, based on received Energy Assignments and internal coordination efforts. An Allocation is delivered to the Resource Manager to control the device.

User Energy Target	Energy targets set by the Prosumer for its energy consumption/generation/storage appliances. This can be financial (optimize for lowest cost or highest gains), infrastructure (optimize for lowest load) or operational (optimize for lowest exchange with net and maximum auto-production consumption). So-called “Static” targets are the default settings, profiles and schedules; “dynamic” targets are ad-hoc deviations from the static targets.

Energy Proposals and Energy Assignments are generic terms; the protocols and formats are not specified or enforced by the FP Application Infrastructure; energy management applications can use their proprietary protocol and format for communicating proposals and assignments with their energy market participant(s). In contrast, Control Spaces and Allocations are defined and specified by the FP Application Infrastructure; their specification is called the **FP RAI** (FP Runtime Application Interface) specification.

3 FlexiblePower Application Infrastructure Functional Architecture

3.1 Entities

3.1.1 Actors and External Systems

3.1.1.1 External Entities

The following actors and external systems are used in the FP Application Infrastructure design; they are therefore represented as entities. We briefly define the entities with their relation to concepts.

When we introduce functional entities, we distinguish different types[9]: Actor entities (A) that represent a person or organization that acts in/with the system, Logical entities (L) that represent a logical artifact/data type, System entities (S) that represents a system or system component.

T	Entity	Description
A	DSO	The DSO entity models the *DSO* concept.
L	DISTRIBUTIONNET	This entity models the distribution system's network (the *Distribution Grid*) concept
L	PROSUMER	This entity models the *Prosumer* concept.
A	SUPPLIER	This entity models the *Supplier* concept, but only in its role as supplier. When the Supplier also acts as a player in the *Wholesale Market*, we model its market function then as a separate ENERGYMARKETPARTICIPANT entity.
A	ENERGYMARKET PARTICIPANT	This entity models an actor in the *Wholesale Market*. • This can be a player in the *Wholesale Market*, such as a Supplier. • This could also be an aggregator, which operates for a group of small consumers. These small consumers can monetize their flexibility and the aggregator can use this grouped flexibility to trade on the wholesale market or provide a service to the DSO. • This could also be the PowerMatcher Auctioneer for example.
A	THIRD PARTY INFORMATION PROVIDER	This entity models the *Third Party Information Provider* concept.

- A PROSUMER receives physical electricity from the DSO via a physical connection with its DISTRIBUTIONNET. The PROSUMER receives commodity electricity from the Supplier.
  - A PROSUMER is only connected to one DISTRIBUTIONNET and has only one DSO.
  - A DSO has many prosumers connected to its DISTRIBUTIONNET.
  - A PROSUMER can have multiple SUPPLIERS, but only one supplier for a given timeslot.
- An ENERGYMARKETPARTICIPANT gives incentives/signals to the Prosumer for SDM (Supply and Demand Management).
  - There can be many ENERGYMARKETPARTICIPANTS interacting with the PROSUMER.

The next diagram shows the relations between these entities.

[9] We indicate the types in the first column of the tables where entities are introduced.

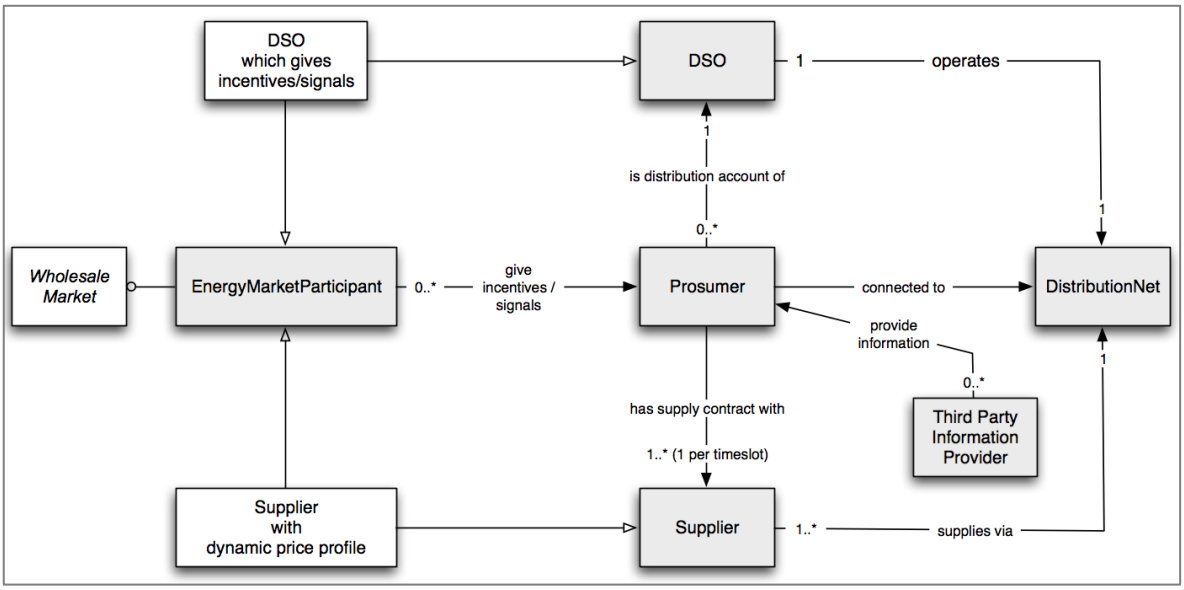


Figure 3-1 Prosumer and External entities

3.1.1.2 *Separate entities by function*

If the SUPPLIER also acts with some dynamics, for example giving incentives/signals with a dynamic price profile, we model the dynamic function of the supplier as a separate ENERGYMARKETPARTICIPANT entity.

A large industrial consumer can buy its energy directly on the *Wholesale Market*. In that case the wholesale market is represented as the SUPPLIER.

The DSO could also give incentives/signals to the PROSUMER; therefore the DSO can also play an active role. Similar to the approach taken with a dynamic supplier, we model the dynamic function of the DSO in that case as a separate ENERGYMARKETPARTICIPANT as well.

3.1.1.3 *Autoproduction*

In the case of so-called autoproduction, the Prosumer has on-site generation and acts as a *DG Operator*. However we do not model this function separately, it coincides with the PROSUMER entity.

3.1.2 *FP Application Infrastructure Systems and Actors*

3.1.2.1 *Internal Entities*

The following FP Application Infrastructure actors and systems are used in the FP Application Infrastructure design; they are therefore represented as entities. We define them with their correspondence to concepts.

T	Entity	Description
L	ACCOUNT	A PROSUMER that participates in the FP Application Infrastructure ecosystem is considered an FP Application Infrastructure ACCOUNT. It represents a single legal entity.
L	DEVICE[10]	A DEVICE is a physical appliance of the PROSUMER that consumes, generates or stores energy. Note that DEVICES, which are monitored/controlled by the FP Application Infrastructure platform, will be represented in the system as RESOURCES (see

[10] A Device is not an entity but a logical concept that refers to a physical device. They are not represented in the system, but abstracted by a Resource representation that communicates with a DeviceProtocol.

		further). The physical appliances that are monitored/controlled (and thus represented as Resources), together with the appliances that are not monitored/controlled (and which have no representation) are all called DEVICES. Examples of DEVICES are smart meters, micro-CHP (micro combined heat and power), household appliances (e.g. washing machine or dish washer that allows shifting the start moment, automatic vacuum cleaner that needs charging), energy storage devices, PV (Photovoltaic) panels, etc.
L	LOCATION[11]	An ACCOUNT has a primary LOCATION, which refers to the physical location such as the house of a household or main/central building of a campus. It can also have multiple secondary LOCATIONS.
S	FP NODE (NODE)	The FP Application Infrastructure platform has an FP Node installed at each of its ACCOUNTS. This NODE is a software system deployed on dedicated hardware (the FP HomeBox) at the primary LOCATION of the account. A NODE is responsible for • Monitoring and controlling prosumer DEVICES • Coordination of generation and consumption of the (DEVICES of the) ACCOUNT. • Communication with third parties, such as ENERGYMARKETPARTICIPANTS, THIRDPARTYINFORMATIONPROVIDERS, SUPPLIERS and the DSO. The NODE monitors and controls DEVICES at all LOCATIONS of the ACCOUNT. In case the ACCOUNT also has secondary LOCATIONS with DEVICES that are not in close enough proximity for specific protocols to communicate with them, one can deploy secondary NODES. The secondary NODES communicate with the primary NODE, so that the primary NODE can monitor and control all ACCOUNT DEVICES, including the remote ones.
S	FP MANAGEMENTCENTER (FP MC OR MC)	The FP Application Infrastructure platform has an FP ManagementCenter (MC) to remotely manage the FP NODES. The MC is a software system, operated by (persons of its) FP MANAGEMENTCENTEROPERATOR
A	FP MANAGEMENTCENTER OPERATOR (MCO)	The FP MANAGEMENTCENTEROPERATOR (MCO) represents the (persons of the) organization that manages the FP Application Infrastructure platform for a certain region, being the region of the DistributionNet. The MCO operates the MC to manage its ACCOUNTS, the MC provides the MCO with remote management of the NODE(S) of its ACCOUNTS.
S	FP APPSTORE (FP AS OR AS)	The FP APPSTORE (AS) is an online store where accounts can buy additional applications to enrich the functionality of their Node. These so-called APPS (see further) are installed in the NODE.
A	FP APPSTORE OPERATOR (ASO)	The FP APPSTOREOPERATOR (ASO) represents the (persons of the) organization that operates the AS of the FP Application Infrastructure platform for a certain region, being the region of the DistributionNet. The ASO operates the AS.
L	FP NET	With the FP NET we identify the entire instance of the FP Application Infrastructure platform for the region. The ACCOUNTS of the FP Net are PROSUMERS connected to the DISTRIBUTIONNET of a DSO, the nodes of the ACCOUNTS are managed by the MC which is operated by the MCO, the AS is the store available to the ACCOUNTS, operated by the ASO.
A	USER	With a USER entity, we identify a person of the ACCOUNT that can interact with

[11] A Location is not an entity but a logical concept that refers to a physical location (house of household or building) that is a grouping of devices in near proximity. A Location therefore does not necessarily require an explicit representation since it already is implicit by the Node controlling/monitoring devices in close proximity using DeviceProtocols.

		the FP NODE of the ACCOUNT. The USER can also interact directly with the physical DEVICES of its LOCATION(S). An ACCOUNT can have multiple USERS, with different roles and access rights (see further).
L	USERDEVICE	A USER (with appropriate access rights) can interact with a NODE via a graphical user interface (GUI) on a USERDEVICE, which can be a tablet, smartphone or PC. As will be discussed later in this design, the GUI is either offered by an application on the USERDEVICE, where the application communicates with the NODE, or the GUI is offered by the Node itself and consumed on the USERDEVICE by a web browser. A USERDEVICE can also supply sensor information to the NODE. This either from the USERDEVICE of the USER interacting with the GUI, or from USERDEVICES not used in GUI interaction. An example of the latter case is the supply of location information from the smartphones of all account users to help the energy management.
L	RESOURCE	A RESOURCE is the representation of a DEVICE in the NODE of an ACCOUNT.
L	DEVICEPROTOCOL	RESOURCES of a NODE communicate with their physical DEVICE using a DEVICEPROTOCOL. Examples of DeviceProtocols are ZigBee, PlugWise, PLC, etc. When DEVICES of a secondary LOCATION are not in close proximity to communicate to with the DEVICEPROTOCOLS, a secondary NODE is deployed at that remote LOCATION to communicate with these remote DEVICES. The secondary NODE communicates with the primary NODE so that these remote DEVICES can also be represented by and controlled/monitored by the primary NODE.

The next diagram shows the entities with their relations:

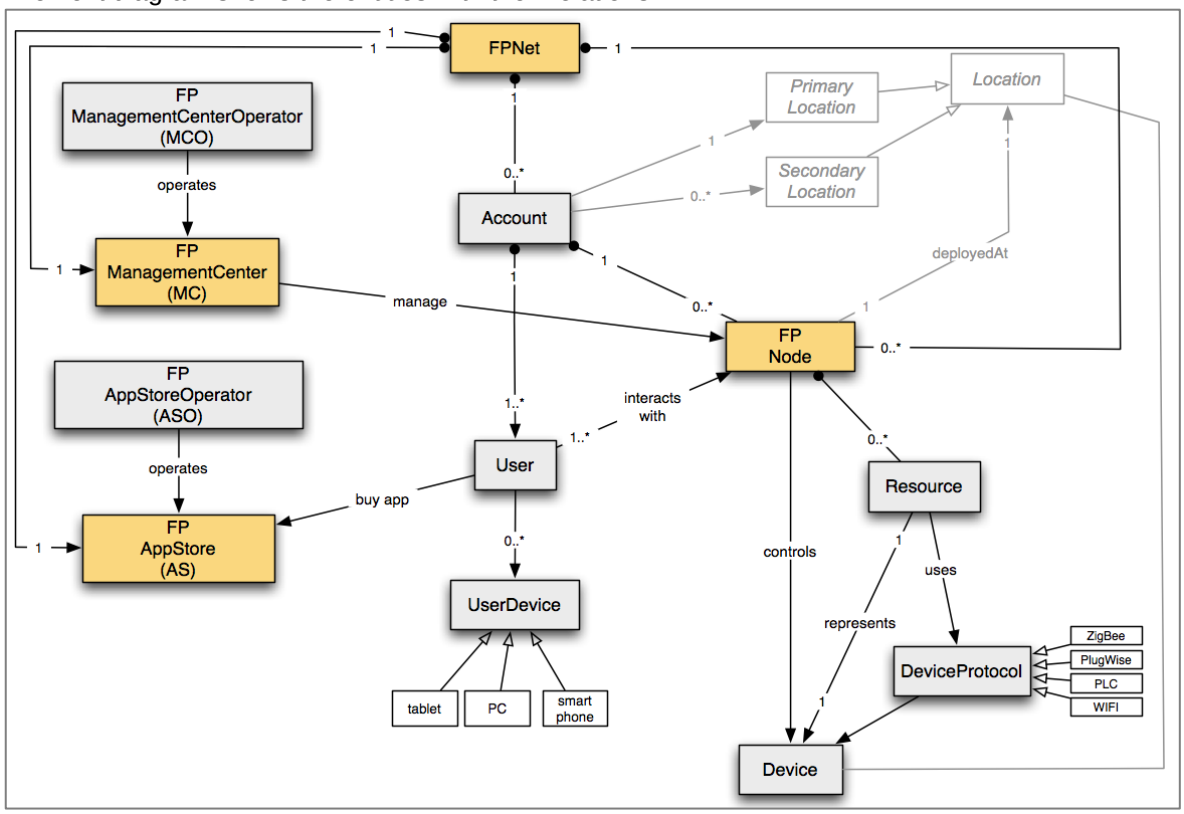


Figure 3-2 Internal Entities

3.1.2.2 *Relations between Entities in an FP Net*

The next diagram illustrates the interplay between different entities.

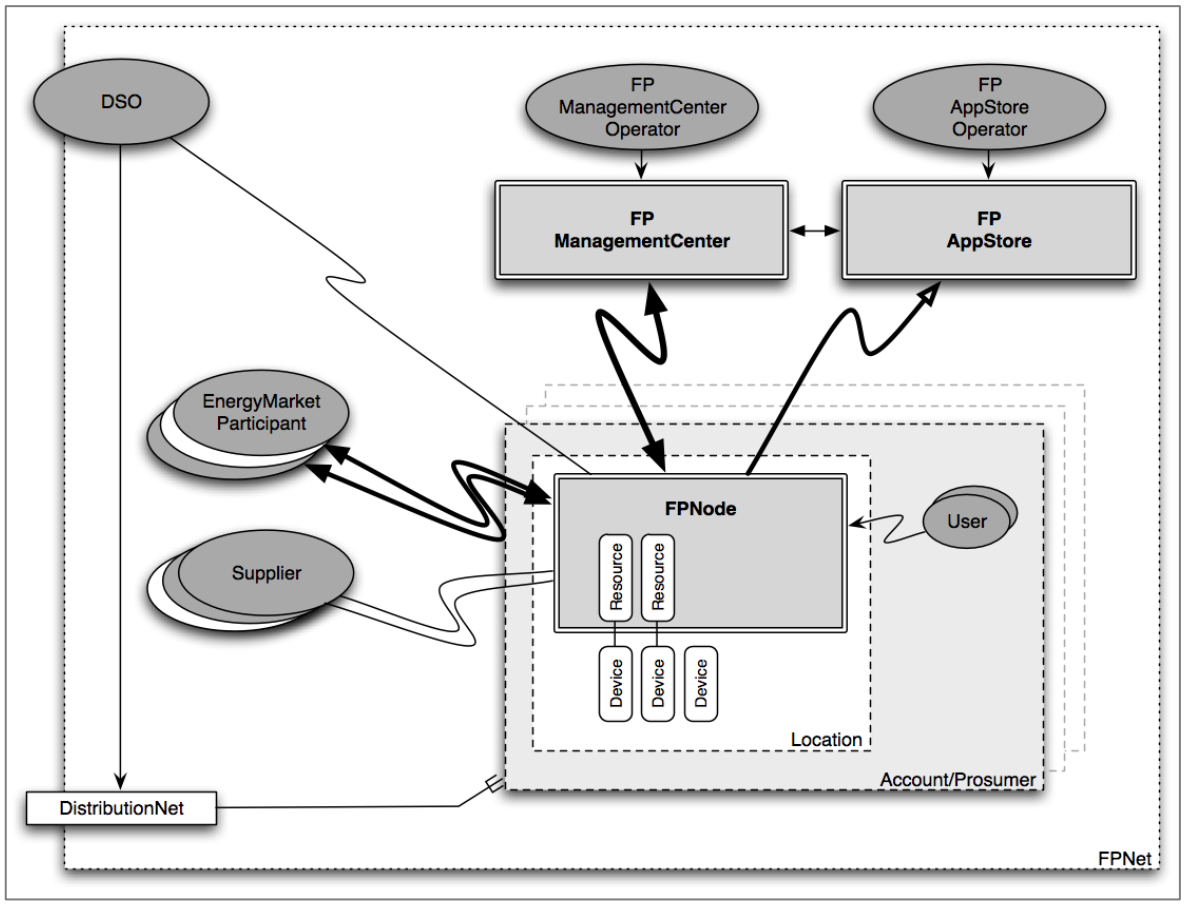


Figure 3-3 An FP Net

- An FP Net has ACCOUNTS, which are PROSUMERS from the same DISTRIBUTIONNET of the same DSO. There can also be PROSUMERS that are no ACCOUNTS of course, since not everyone is obliged to participate.
- Each ACCOUNT has an FP Node deployed at its primary LOCATION.
- Devices at the LOCATION are controlled/monitored by RESOURCES in the Node.
- There is a single MC managing all nodes of the FP NET. The MC is operated by the MCO.
- There is a single AS where Accounts can buy Apps for their NODE. The AS is operated by the ASO.
- An ACCOUNT and therefore its NODE can interact with multiple ENERGYMARKETPARTICIPANTS.
- An ACCOUNT and therefore its NODE can interact with the SUPPLIER. There can be multiple SUPPLIERS for a single NODE, but only one SUPPLIER per given timeslot.
- There can be multiple THIRDPARTYINFORMATIONPROVIDERS that supply information to the node.
- All ACCOUNTS are connected with the same DISTRIBUTIONNET.
- An ACCOUNT can have multiple USERS, interacting with the NODE of the ACCOUNT (and directly with its DEVICES).

3.1.2.3 *When an Account has multiple Locations*

In the case of a multi-location Account, one can observe multiple Nodes.

- There is a main FP NODE deployed at the primary LOCATION.
- Other NODES at secondary LOCATIONS communicate with their primary NODE, this to provide the primary NODE with all information to also monitor/control the remote DEVICES. In this way, the primary NODE views all RESOURCES and is able to control/monitor all DEVICES in all LOCATIONS of the ACCOUNT.
- All major communication with the MC, ENERGYMARKETPARTICIPANTS, DSO and SUPPLIERS, is performed by the primary NODE. There is minor management communication between secondary NODES and the MC.
- The USERS interact with the primary NODE. Since the primary NODE can communicate with the secondary NODES, USER interactions can also influence secondary NODES.

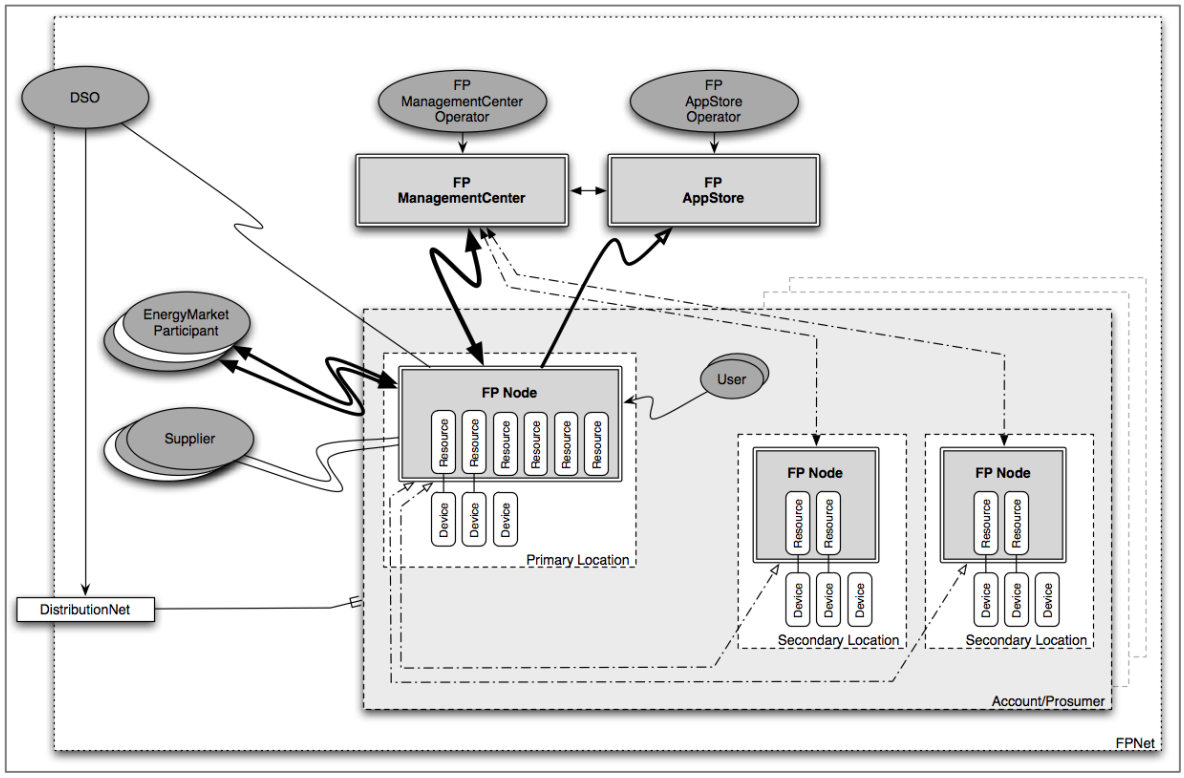


Figure 3-4 An FP Net with a multi-location account

3.1.2.4 *Multiple FP Nets*

The FP Application Infrastructure platform supports multiple FP NET instances.

An FP Net, which is about PROSUMERS of the same DISTRIBUTIONNET, has SUPPLIERS and ENERGYMARKETPARTICIPANTS that in reality are probably organizations that also act as SUPPLIER or ENERGYMARKETPARTICIPANTS in other FP NETS. However we do not model these as shared entities, for the FP Application Infrastructure platform we consider them to be different entities, even if in reality they would be the same organization.

On the other hand, the design takes into account a possible hierarchy of FP MANAGEMENTCENTERS and a hierarchy of FP APPSTORES.

We have an FP Net with its FP ManagementCenter and its FP AppStore by DistributionNet/DSO, similar to having an iTunes application store by nation (due to legal differences by nation on music property and distribution rights).

- It is considered useful to have a hierarchy of stores to allow for apps to have a wide market beyond one DistributionNet.
- Since there is a close interaction between the AS and the MC of an FP Net, it is considered useful to still have an AS for each FP Net instead of having a store common for all FP Nets.
- The design includes the possibility of having a hierarchy of management centers and stores. This provides the most flexibility in organization, avoiding duplication of effort and process, while still preserving all advantages of local management centers and stores.

Since a MC manages an FP Net, which has a DistributionNet and DSO as its domain, a management center higher in the hierarchy is not a real management center since it spans different DistributionNets. We rather call this an FPGroupManagementCenter (GMC). Similarly we have an FPGroupAppStore (GAS).

T	Entity	Description
L	FP GroupManagementCenter (GMC)	We can have a hierarchy of management centers. A GMC manages either different MC's, or different GMC's. Typically one could have a GMC for a transition network that manages different MCs for the distribution networks. A GMC higher-up in the hierarchy could be for a national level, etc.
L	FP GroupAppStore (GAS)	We can have a hierarchy of application stores. A GAS manages either different AS's, or different GAS's.

Note that these group entities are not necessarily part of the software model for the FP Application Infrastructure, since they fall outside the scope of a single FP Net. It is only when the FP Application Infrastructure software implementation would have a scope extending a single FP Net that these entities become important to distribute and share information, responsibilities and process.

The next diagram illustrates the entities and their relations:

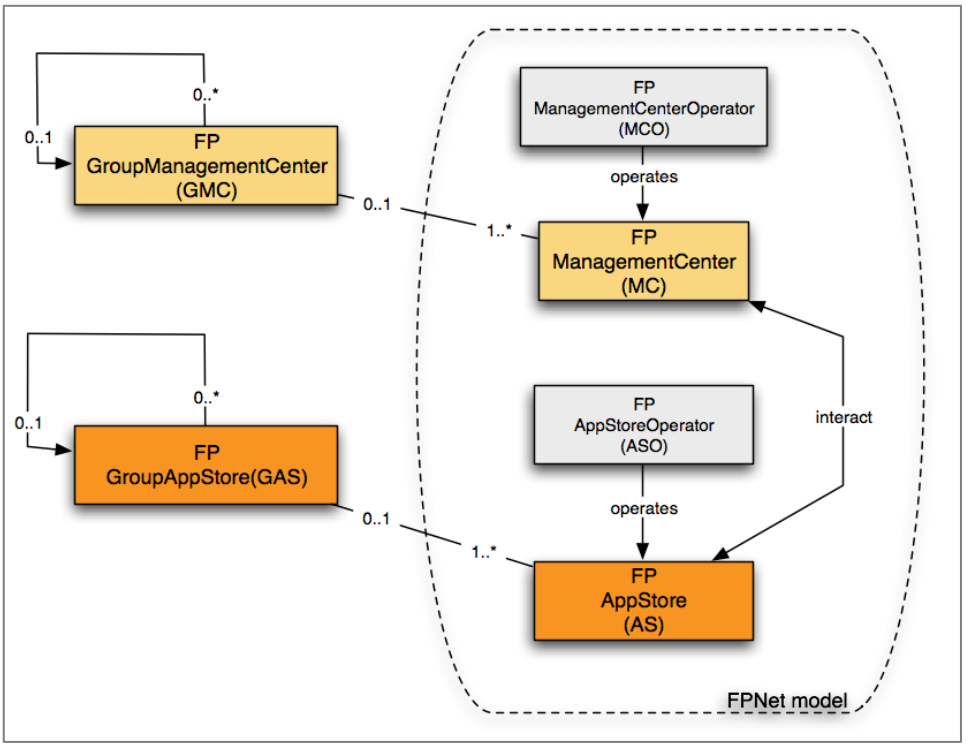


Figure 3-5 Group management centers and stores

The next diagrams depicts two FP Nets where one can see that each FP Net has its own MC and AS; the MC and the AS of an FP Net have interactions; the MC of an FP Net can belong to a GMC; the AS of an FP Net can belong to a GAS (not shown in the diagram).

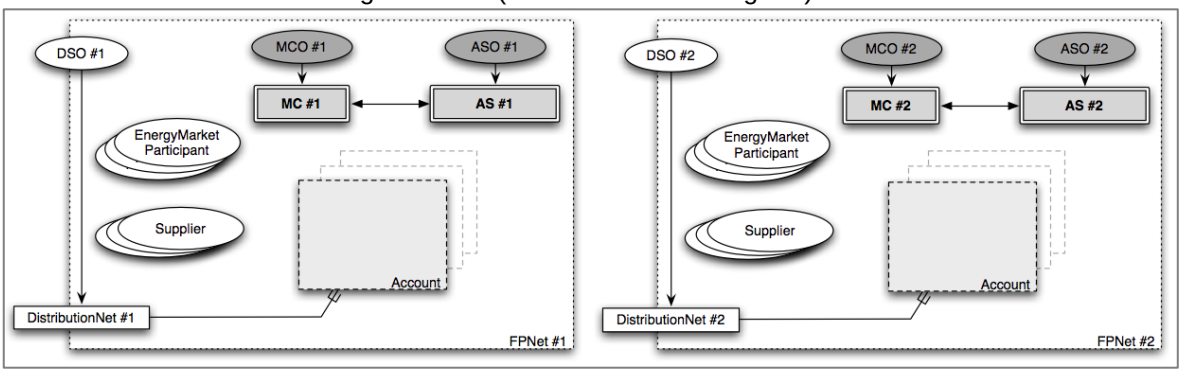


Figure 3-6 Two FP Nets

3.1.2.5 *Separate entities by function*

If the DSO would also act as the MCO of the MC of its FP NET, we still represent the DSO and the MCO separately. If the DSO would also act as the ASO of the AS of its FP NET, we still represent the DSO and the ASO separately. In the case that the DSO would also act both as MCO and ASO, we still have three separate entities to separate the functions: the DSO, the MCO and the ASO.

If an ASO would operate multiple stores, we continue to represent each ASO of each AS separately. If a MCO would operate multiple management centers, we represent each MCO of each MC separately.

The other parties involved in an FP Net, such as ENERGYMARKETPARTICIPANTS, Suppliers, DSO and THIRD PARTY INFORMATION PROVIDERS, all have separate representations for each FP NET, even it some of them would correspond to the same organization.

3.1.2.6 *System entities and their deployment*

From a functional point of view we have three types of system entities in an FP Net:

- An FP Net has at least one Node for each of its Accounts. Each Node is deployed at the Location of the Account. In the case of a multi-location Account, all of its Nodes are deployed at the Locations (one primary and optionally multiple secondary) of the Account.
- An FP Net has one MC. The MC of an FP Net is deployed typically at the site of the MCO, at a data center or hosting facility accessed by the MCO, or in the cloud.
- An FP Net has one AS. The AS of an FP Net is deployed typically at the site of the ASO, or at a data center or hosting facility accessed by the ASO, or in the cloud.

3.1.2.7 *Communication*

The next diagram shows the main entities, it highlights the system entities (orange) and also the partners that communicate with the node of an account.

Note that DSO, SUPPLIER(S), ENERGY MARKET PARTICIPANTS and THIRD PARTY INFORMATION PROVIDERS are all partners that can communicate with the node of an account. Their detailed interactions are described further in this document.

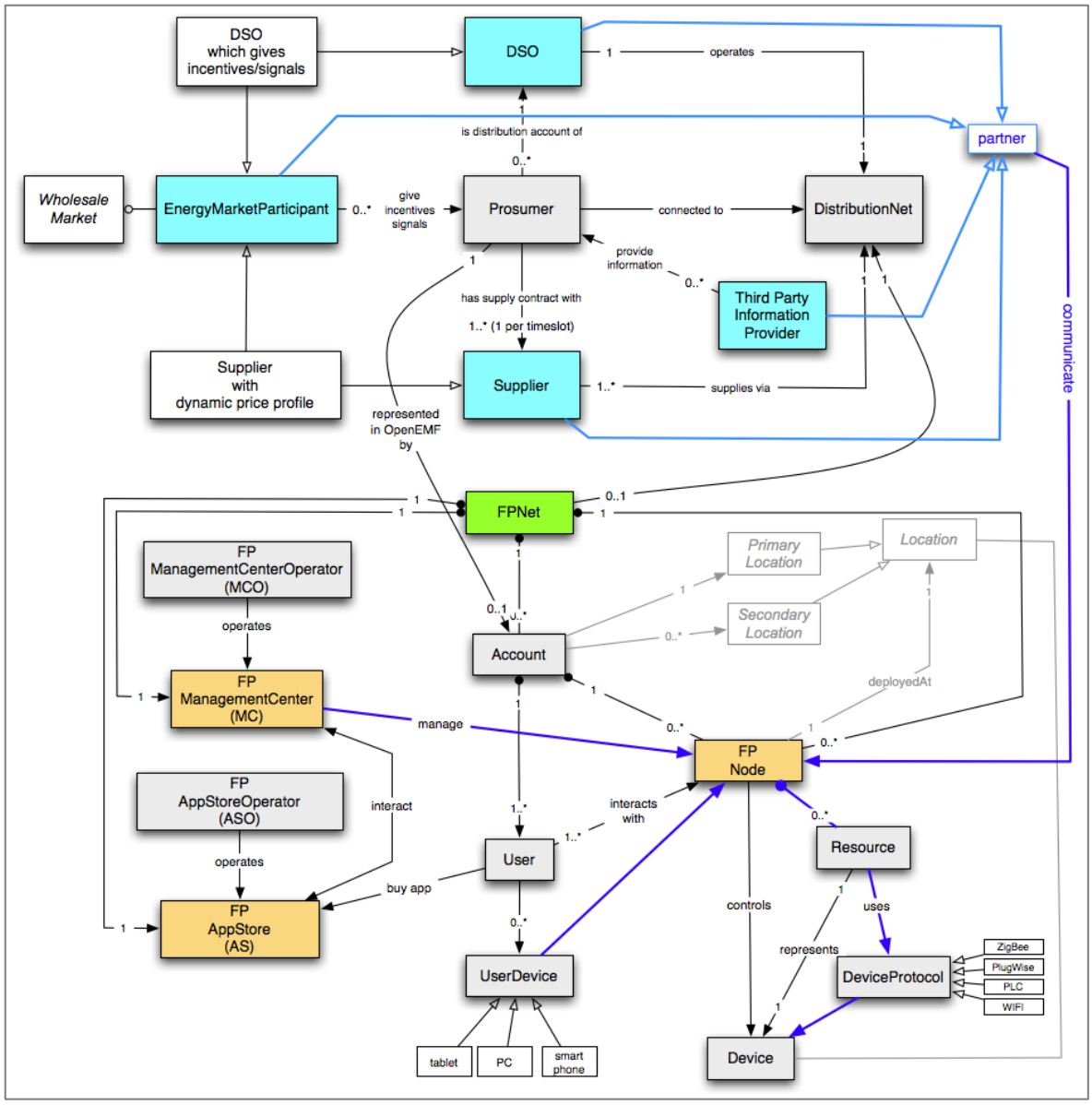


Figure 3-7 Entities, system entities and node communication partners

3.1.3 *Component Entities*

An FP Node has much functionality and is organized in a number of components. Besides the technical decomposition, there are a number of components identified in the functional design because they have specific attributes concerning

- being required or optional components
- having the ability to be installed, updated and/or replaced in a live system
- being standard or being purchased

The FP Node is a plug-in architecture where we use the following terminology[12] about components:

Term	*Description*
software	This is the FlexiblePower Application Infrastructure platform software, without the underlying firmware (see below).
firmware	This is the lower-level software on top of which the FP Node software runs. Depending on the technical architecture and design, this could include for example: the OS, the Java runtime environment, OSGi platform provisions and TR-069 remote management software.
component	A *component* is a constituent of the *software*.
bundle	A *bundle* is a *component* that can be installed, updated and replaced into the FP Node system on a live system, meaning without requiring a restart of the *FP Node software*.

We define the following entities:

Entity	*Description*
PLUGIN	A PLUGIN is a *bundle* that is required and that is provided by the management center as default.
APP	An APP is a *bundle* that is optional and needs to be purchased in the store. It can be an optional additional bundle, or it can be an optional replacement for a PLUGIN.

The next table lists the attributes for each of the types:

Component / attributes:	*Required / Optional*	*Standard / Purchased*	*Install / update / replace*
component	*	*	*
bundle[13]	*	*	Live
PLUGIN	Required	Standard	
APP	Optional (Additional)	Purchased	
	Optional (Replacement for a Plugin)		

Some APPS have dependencies and need to be installed together.

Entity	Description
APPGROUP	An APPGROUP is a group of Apps that have a dependency on each other. They are typically

[12] Please note that the terminology introduced here is of a functional nature only. In this functional view we identify all options for extending the functionality of the FP Runtime as being individual Apps, which can be grouped in App Groups. However, there are other perspectives where this is entirely different. From a User perspective, an App provides additional functionality and has at least one visual component, but the underlying composition is not known by the user. Also, in the technical realization the terminology is quite different. The reference implementation 12.11 defines an App as a collection of modules (which are jars/bundles) which can provide factories to instantiate ConfigurableServices of which typical types are resource managers, resource drivers etc. They can also register Widgets that provide a GUI aspect. So in this technical realization, a technical App corresponds to a functional App Group, a technical App module corresponds to a functional App, more correctly a factory from which one can create multiple running apps and visual widgets.

[13] In this context “bundle” is a functional term, not necessarily corresponding with a technical OSGi bundle.

	offered together in the store and need to be installed together to operate correctly.

The FP Node with its firmware and software, and its components including bundles and apps, is typically deployed on an FP HomeBox. There are however a number of variants that will be described further in this design document.

The next diagram shows the component entities and their relation with the constituents of the FP Node. On the left in the diagram we show that the FP Node can have different functions and deployments. The most typical is having one FP HomeBox in the premise of the account; that FP HomeBox runs an FP Node as single and thus primary Node. When however an account has multiple locations that a typical nearby device protocol cannot cover, one can have multiple FP HomeBoxes. One runs the FP Node as a primary node, others run FP Node software in the function of secondary nodes, communicating the control spaces and allocations of devices in more remote locations to the primary node. Another situation exists when the FP Node is not deployed on a local FP HomeBox, but on a remote server. In that case, the FP Node functions as an FP Remote Node, connecting to device protocols via the home area network.[14] To make abstraction of the specific deployment and function we mostly refer to the FP Node instead of specifically referring to the FP HomeBox.

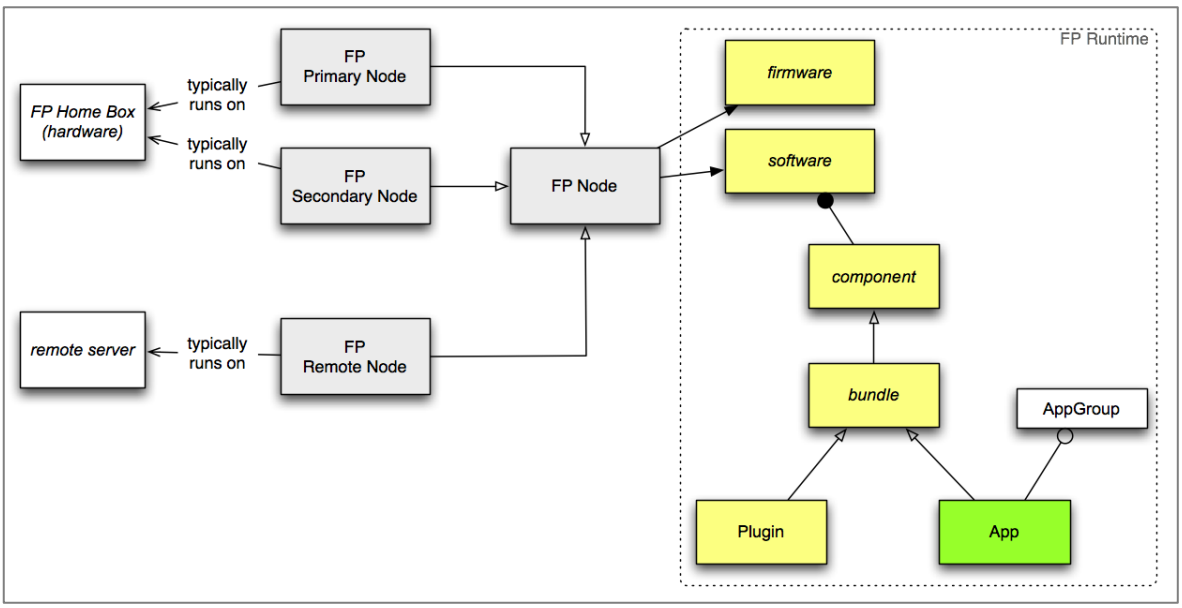


Figure 3-8 FP Node component entities

Referring to the concepts of the functional design, we already can identify components of different kinds:

- Some components of the FP Node, such as the provisions to ensure security and local management of the node are *components*, but not necessarily *bundles*.
- Energy Applications are typically Apps. However rolling out the FlexiblePower Application Infrastructure with the FlexiblePower Application pre-installed would make that application a PLUGIN.
- Resource Managers and Resource Drivers are typically APPS, since they are optional components that are purchased to control/monitor a specific device. They typically belong

[14] There is even an additional situation not shown in the diagram. This is the case where some of the FP Node components are embedded on devices. When appliances have embedded resource manager and resource drivers, they communicate their control spaces and receive allocations to/from the FP Node software on the FP HomeBox.

to the same APPGROUP, both need to be installed to control the device. Such an APPGROUP could even include additional Apps, as explained later.[15]

3.1.4 *Provision of system, plugins and apps*

The FP ManagementCenter is the provider of

- The FP Node hardware (FP HomeBoxes) and initial firmware and software.
  - The provision is realized by the physical installation of the Node(s) at the location(s) of the Account.
- Updates of FP Node firmware.
  - Such an update is allowed to require a restart of the FP Node
- Updates of the FP Node software (the system)
  - Such a system update is allowed to require a restart of the FP Node
- Plugins and their updates.
  - A Plugin is a bundle and therefore does not require a restart to install an update.
- Apps and their updates.
  - An App is an optional bundle and therefore does not require a restart to be installed or to be updated.

All APPS are offered for purchase by the FP AppStore (AS), but their provision/download is handled by the FP ManagementCenter. All optional bundles are APPS, all APPS are offered by the AS and the AS only, including the free optional bundles originating from the MCO or the FP Application Infrastructure platform in general.

- ➔ Thereby the AS is the only entity in the FP NET where an ACCOUNT USER can choose for an optional bundle.
- ➔ Thereby the MC is the only entity in the FP Net where software for the FP Node (firmware, software system, PLUGINS, free APPS, commercial APPS) is downloaded from.

3.1.5 *Identifications, categories and information feeds for Apps and Plugins*

Replaceable components (Apps and Plugins) have a category identifying their function and deployment, a Guid (Global Unique Identifier) to uniquely identify them and optional attributes if they are partner of a third party information provider; by being a destination or subscriber to information feeds.

[15] An example of this will be an App deployed in the INTERACTIONFRAMEWORK of the node to extend the GUI with a GUI page for the User to have a specialized view and control instrument for the specific device. This App will communicate with the ResourceManager supplying the GUI page or the specific information to build the GUI page. Such an App is then a third App belonging to that same AppGroup.

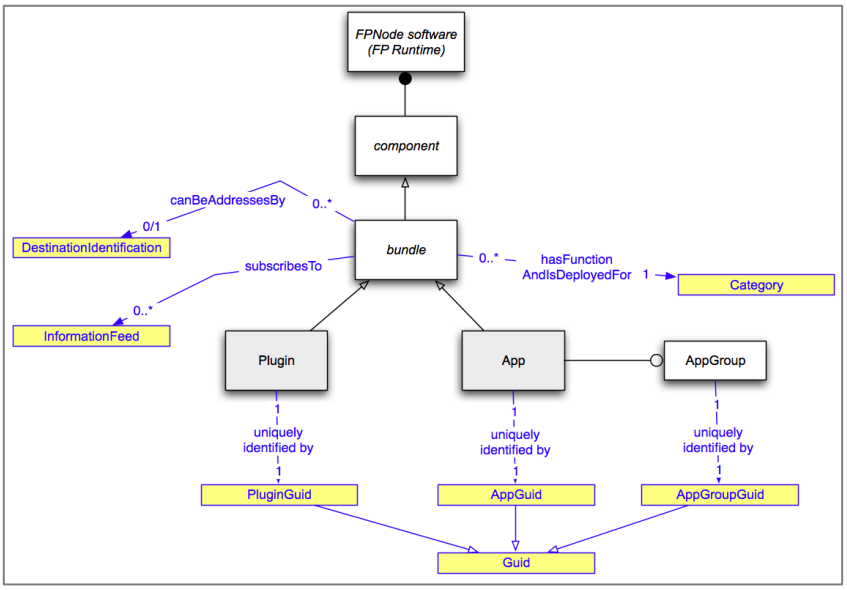


Figure 3-9 Guids, Categories, Information destinations and feeds

3.1.5.1 *Category*

Apps and Plugins have a category, which determines where they can be deployed. Categories include for example: ResourceManager App, ResourceManager Helper App, EnergyApp, etc. All categories will be described in chapter 4.

3.1.5.2 *Guid (Global Unique Identifier)*

Plugins and Apps must have a unique identification in an FP Net to identify them.
An AppGuid is an App identifier unique in the FP Net for that specific version of the App.
An AppGuid either identifies an individual App version, or identifies an AppGroup containing multiple Apps (which again have AppGuids).

3.1.5.3 *DestinationIdentification*

Apps and Plugins that want to consume information from a Third Party Information Provider, for example a server-side part of an App supplying weather information to a ResourceManager helper App, have to label themselves with a so-called DestinationIdentification. The provider can then supply information and include that DestinationIdentification so that the information is delivered to the App. Note that the DestinationIdentification can be broader than the Guid if one wants to include multiple versions of an App in the destination identification.

3.1.5.4 *InformationFeed*

Apps and Plugins can specify InformationFeed labels to express that they want to subscribe to general information feeds. Such information feed is defined by the FP Application Infrastructure platform, with a specific format to specify the information.

As an example, the FP Application Infrastructure platform could define an information feed category “weather forecasting info” and define a format for the information.
ThirdPartyInformationProviders can then provide weather forecasting information using this format.
Components in the FP Node could use this information by announcing that they subscribe to these feeds. These can be plugins or Apps. The security and local management infrastructure of the node knows who subscribes to what feed, so it can direct the incoming information feeds to all components that have a subscription.

3.1.6 *Security and Management Entities*

There are a number of functional entities (which one also could call information types) that are important for the organization of management and security in an FP Net.

Entity	Description
CERTIFICATE	A CERTIFICATE in the FP Application Infrastructure platform is an electronic document to establish the identity of a party. Examples are partner-certificates that are required for the FP NODE to communicate securely with another party. The certificate establishes that the party is a trusted partner. The FP MANAGEMENTCENTER issues certificates; it is in other words the certificate authority. A CERTIFICATE is valid for a time period. One can ask the certificate authority to renew a certificate to extend the time period. The certificate authority can revoke a certificate prior to the expiration of the certificate time period.
AUTHORIZATION	An AUTHORIZATION in the FP Application Infrastructure platform is the granted right to execute an action or to access resources. In the FP Net, the FP MANAGEMENTCENTER performs global authorization; the *Security Management Framework* of the FP NODE provides local authorization, based on global authorizations obtained by the MC. For example, an FP Node needs to ask authorization to the MC to use an APP. Once the authorization is obtained, the security framework of the NODE can manage the authorizations on a local level. As another example, a node needs authorization to communicate with a partner, for example an ENERGYMARKETPARTICIPANT. The *Security Framework* of the NODE will ask authorization by asking the MC for the partner certificate information and will verify the certificate provided by the partner upon communication.
RIGHT	A RIGHT is an authorization of a set of actions that an entity is allowed to perform. A USER of an ACCOUNT can typically have administration rights, the right to buy and install apps, the right to set static user targets for energy consumption/generation devices, the right to change dynamic device settings, etc.[16]
ROLE	A ROLE is collection of RIGHTS assigned to an entity. A USER of an ACCOUNT has one or more ROLES. Typical roles are <administrator> (rights to administer the FP NODE, add USERS, change USER ROLES, configure DEVICEPROTOCOLS, etc.), <operator> (rights to add DEVICES, buy APPS, set static energy targets of devices, etc.), <regularConsumer> (right to use consumption devices and change their dynamic targets, etc.). An ACCOUNT has one or more ROLES, being the collection of ROLES that their USERS can be granted by the administrator (which is a USER having the <administrator> ROLE).
ACCOUNTTYPE	An ACCOUNT has an ACCOUNTTYPE, which determines the initial ROLES of the ACCOUNT. One could typically differentiate between large industrial prosumers and regular households.
ENERGYTYPE	An ACCOUNT has an ENERGYTYPE that determines its initial energy consumption and generation targets. This is typically used to ensure that new ACCOUNTS can have initial settings, which are favorable for their energy profile. For example having appropriate initial static energy targets ensures that they have a favorable profile already in the initial period, prior to the history information buildup that feeds forecasting and learning models in the system.
POLICY	A POLICY is a set of rules and parameters that guides the operation of components and functions of the FP NODE. The *Security Management Framework* of the NODE ensures that the NODE complies with the POLICIES. The FP MANAGEMENTCENTER determines the POLICIES that the NODE(s) of the ACCOUNT has to comply with. POLICIES can be changed by the MC and should be updated and applied by the NODE.

[16] Note that “static” targets for a device are typically default targets, “dynamic” targets are occasional deviations from those default targets.

The next diagram shows the relations of these security and management entities with the other entities. Specific instances of ROLES and RIGHTS in this diagram are for illustration purposes only.

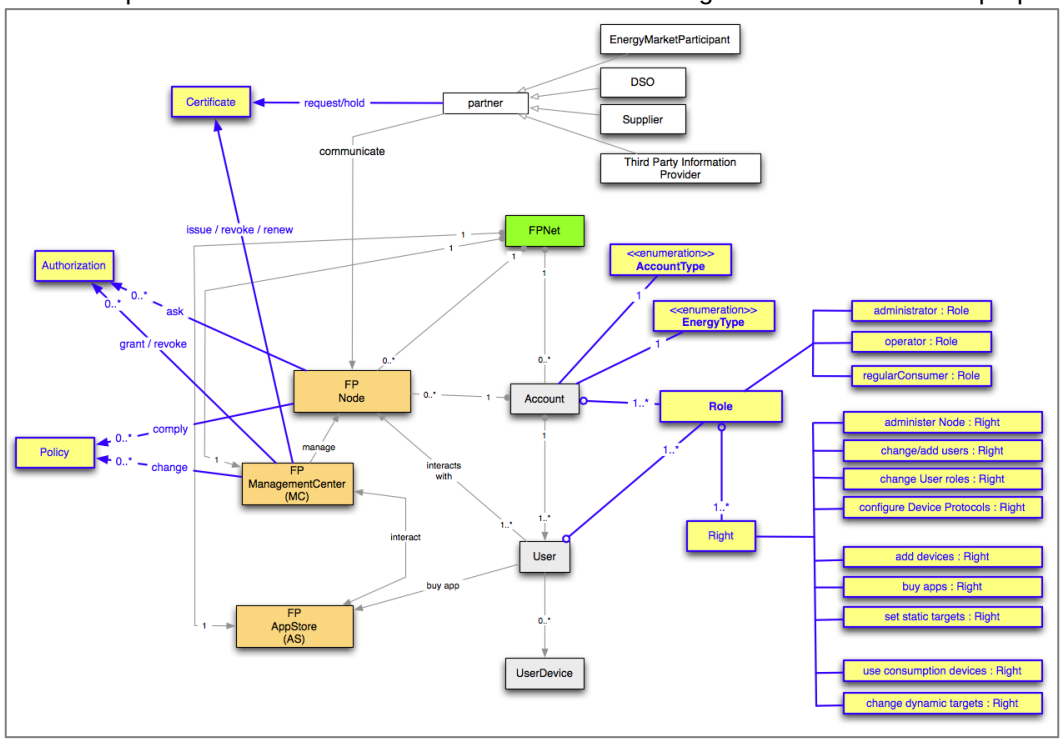


Figure 3-10 Security and Management Entities

Note that “Permission“ entities are not introduced here, since they are more enabling artifacts to realize the security framework. *Rights* are rules that regulate which user or entity can perform *tasks* (such as installing apps, change energy targets, add users, etc.), whereas *Permissions* are rules that regulate which users or entities can use a *resource* (such as a network resource). Since the design has a functional focus on users having rights for tasks, there is no need to introduce permissions. A technical realization however will employ permissions, not only to express user rights management but also to regulate access of components to other components or resources.[17]

[17] The reference implementation 12.11 uses the OSGi permissions to enforce security aspects. Apps (their constituent modules) have to declare the required permissions for accessing resources, the provisioning infrastructure and process will grant permissions based on user confirmation, and the infrastructure will enforce permissions. On the other hand the reference implementation also has user rights management to enforce roles for users covering rights to execute tasks.

3.1.7 *FP Node software system entities*

To conclude the overview of high-level entities in the FP Application Infrastructure platform, we introduce software entities that correspond to components in the FP Node software.

These are entities that correspond to general concepts. They are required to specify the overall management, organization and security principles, since they have specific responsibilities and play specific role in the overall interactions.

Entities	Description

SECURITY MANAGEMENT FRAMEWORK (SMF)	The FP NODE has a SECURITYMANAGEMENTFRAMEWORK (SMF), realizing the *Security Management Framework* concept, which has
	• Security responsibilities
	  ○ Guarding the overall security of the Node with respect to its internal operations, its communications to and from other parties, the security of its data and the protection of its connections with devices.
	• Local Management responsibilities
	  ○ Performing local management of the Node with respect to its representations, information flow, activities of its components and communication messages.
	  ○ Exchanging information with the MC to comply with the MC management actions, but also syncing information to the MC for monitoring, analysis and archiving.
	The SMF performs its duties in coordination with the FP MANAGEMENTCENTER.

INTERACTIONFRAMEWORK (IF)	The FP NODE provides a GUI (see *Graphical User Interface* concept) to its Users. This function is realized by its INTERACTIONFRAMEWORK. It either serves the GUI for consumption by a web browser on a USERDEVICE, or it provides information for a client application building the GUI. The INTERACTIONFRAMEWORK also consumes sensor information submitted by USERDEVICES. It can provide information via Web Services. The INTERACTIONFRAMEWORK can be extended by Apps.[18]
ENERGY APPLICATION LAYER (EAL)	The *Energy Application* concept is realized by the ENERGYAPPLICATIONLAYER (EAL) of the Node. The PLUGIN and APPS in this layer provide the decision logic to realize the *Energy Application* concept. An ENERGY APP resides in the ENERGYAPPLICATIONLAYER, it communicates with an EnergyMarketParticipant. There can be multiple Energy Apps present in the EAL. We discuss this in the next paragraph.
RESOURCE ABSTRACTION LAYER (RAL)	The RESOURCEABSTRACTIONLAYER (RAL) of the NODE contains the RESOURCES that control and represent the physical DEVICES. A RESOURCE is realized by a RESOURCEMANAGER and a RESOURCEDRIVER. Note that a RESOURCEMANAGER and a RESOURCEDRIVER are APPS. They belong to an APPGROUP. Prior to further elaborations, also note that a

[18] In the reference implementation 12.11 this is realized by Apps providing Widgets that become part of the overall GUI when they are registered.

	RESOURCEMANAGER APP can on itself have/contain additional APPS. Also note that the RAL can have other type of device-related apps, such as a METER[19]. which supplies information readouts from a smart-meter to provide the FP Runtime with information about the appliance, for example electricity consumption information.
RESOURCEMANAGER	The RESOURCEMANAGER of a RESOURCE is an APP realizing the *Resource Manager* concept. A RESOURCEMANAGER can by itself also have/contain additional APPS. A RESOURCEMANAGER has operational status information of its Device such as temperature. It receives these essential parameter values from the ResourceDriver. It also has device characteristics such as the heating curve. This is part of the RESOURCEMANAGER functionality. It also knows about (dynamic) user energy target settings such as a desired temperature interval. It receives this information via the Device itself or via the GUI (see further). From the operational status, the device characteristics and the energy target settings, the ResourceManager can derive the energetic flexibility of the device.
RESOURCEDRIVER	The RESOURCEDRIVER of a RESOURCE is an APP realizing the *Resource Driver* concept. RESOURCES communicate via their RESOURCEDRIVER to the physical DEVICE using a DEVICEPROTOCOL. In case one succeeds in separating device specifics from the protocol specifics, a RESOURCEDRIVER can consist of a separate DEVICEDRIVER and PROTOCOLDRIVER. In case the separation is not completely realized, these are not separate entities but rather implicit functions.
DEVICEDRIVER	This is the protocol-independent part of the RESOURCEDRIVER.
PROTOCOLDRIVER	This is the protocol-dependent part of the RESOURCEDRIVER. It serves as a protocol convertor translating the device specific instructions into the protocol specific language to communicate with the device via the protocol. Note that the design does not enforce the separation between DEVICEDRIVERS and PROTOCOLDRIVERS, but a separation, when possible, makes the approach more generic and allows reuse.
PROTOCOLCONNECTOR	A PROTOCOLCONNECTOR represents the connection bridge between the Node and the protocol realization itself. For a node to be capable of using a specific protocol, it needs to install a PROTOCOLCONNECTOR. In the case of ZigBee for example, the ProtocolConnector communicates to the ZigBee coordinator (ZC), to reach the ZigBee End Devices (ZED).

[19] A METER supplies data from a smart-meter to provide the FP Runtime with information about an appliance, for example electricity consumption information. Other FP Runtime components such as apps can then use this information.

The next diagram summarizes the FP Node entities.

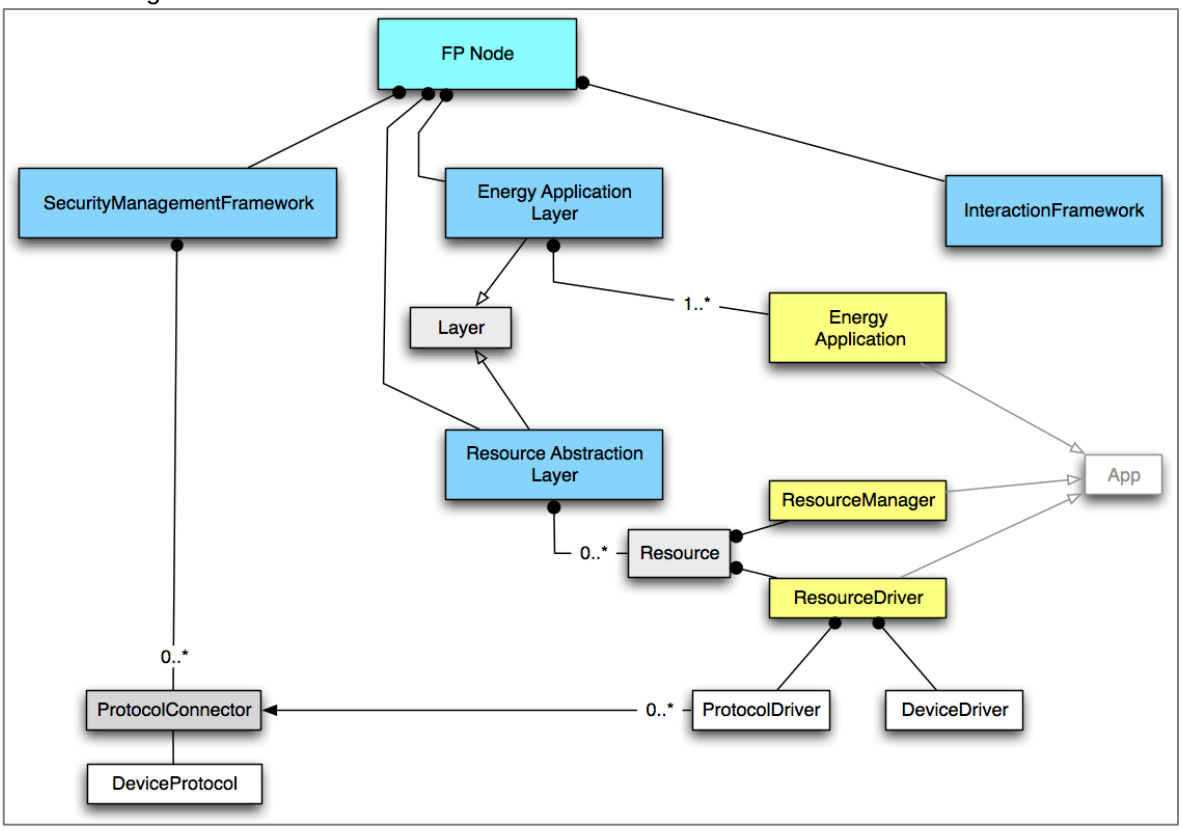


Figure 3-11 FP Node software entities

3.1.8 *Energy Applications interacting with EnergyMarketParticipants and Resources*

We examine in more detail the relations between Energy Applications and Resources, together with the EnergyMarketParticipants and Devices. The next figure illustrates the relations explained below.

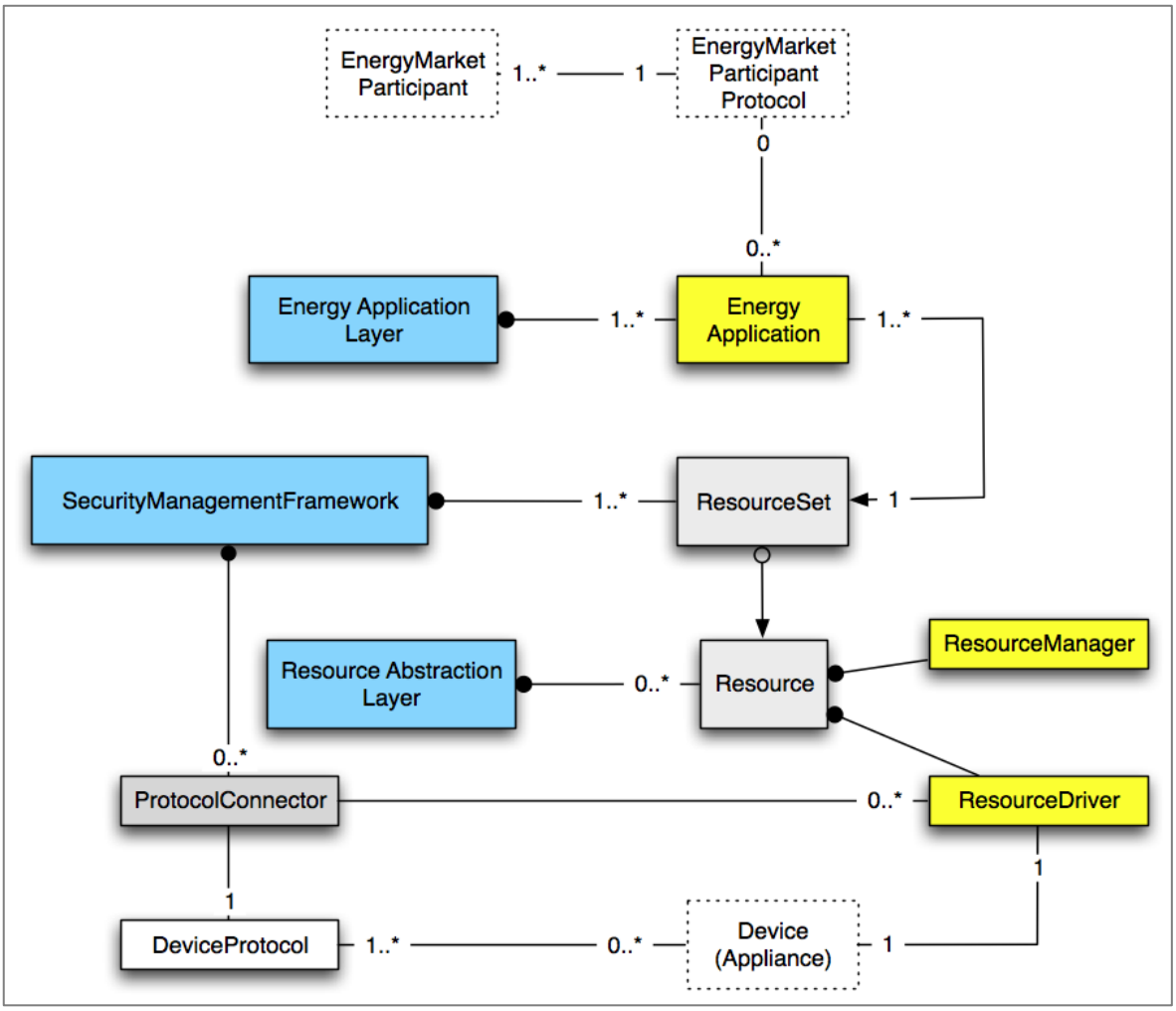


Figure 3-12 Relations of Energy Applications, participants and Resources

The Resource Abstraction Layer can have many Resources, which are abstractions of devices. A ResourceManager and a ResourceDriver represent resources. A ResourceDriver represents 1 Device and communicates to the Device via a DeviceProtocol connected with a ProtocolConnector. ResourceDrivers are Apps; when the same ResourceDriver App is used for two devices, we consider these two ResourceDriver instances, although this is (probably) one App purchase. A ResourceManager is an App, which works with a ResourceDriver. The ResourceManager exposes the energetic flexibility of the device to (an) Energy Application(s).

The Energy Application Layer has at least one Energy Application but it can have multiple. An Energy Application is considered a Plugin, it can however be replaced by an App and there can be multiple.

An Energy Application makes EnergyProposals and receives EnergyAssignments back from an EnergyMarketParticipant.

An Energy Application communicates either with a single EnergyMarketParticipant, with none, or with multiple. The latter only when they use the same protocol given that the protocol has special provisions. The first case is the most common. In the second case, the Energy Application performs its energy management only using the energetic information from the account resources without interacting with the market. In the final case where an Energy

Application communicates with multiple EnergyMarketParticipants using the same protocol, the protocol must have provisions to retract a proposal and to reject an assignment, since there is no assumption that different participants can communicate with each other to reach agreement on their assignments. It is then the responsibility of the Energy Application to inform the other participants if it receives an assignment from one participant that covers an open proposal made to another participant.

An Energy Application is associated with a set of resources, a ResourceSet. If there is no ResourceSet specified for the Energy Application, its ResourceSet contains all Resources of the Account.

In case there are multiple Energy Applications, there are two possibilities. Two Energy Applications that have disjoint ResourceSets work alongside each other without any conflict, since their EnergyProposals are based on disjoint set of ControlSpaces and they can realize EnergyAssignments for these resources. If the ResourceSets of two Energy Applications are not disjoint, then they work in competition. This is allowed as long as the protocols they use for their EnergyMarketParticipants are permissive for not executing assignments or have provisions to communicate back when they will not execute an assignment.

The association between an Energy Application and its ResourceSet can be defined by the User, via the GUI. An Energy App and equivalently a ResourceManager App can have restrictions for which associations are allowed.

Note that so far we only discussed Energy Apps (residing in the Energy Application Layer), and Resource related Apps, such as Resource Manager Apps and Resource Driver Apps (residing in the Resource Abstraction Layer). There are other categories of Apps that will be introduced further in the design document.

3.2 Overall Functional Architecture and Principles

3.2.1 Systems and dependencies

The FP Application Infrastructure platform architecture is designed to offer a high degree of independence between the main entities of the platform.

In the next diagram we show the dependencies between the main entities. Below we describe their nature using the labels from the diagram. Note that we have a single FP ManagementCenter and a single FP AppStore for an FP Net. We have multiple partners such as EnergyMarketParticipants, Suppliers, DSO and ThirdPartyInformationProviders. There is an FP Node for each Account.[20]

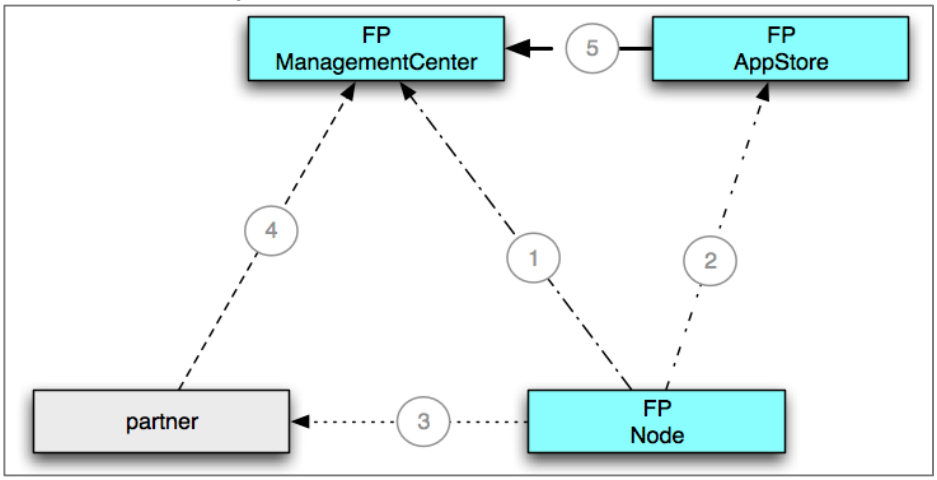


Figure 3-13 Dependencies between entities

1: The Node is *highly independent* from the MC

- When the MC is unavailable, the Node can still continue its operations.
  - The Node continues communication with known partners until their certificates expire
  - It cannot sync its data with the MC, but it keeps everything local until it can sync later
  - Only a few actions that require MC authorization are not possible

2: The Node is *highly independent* from the AS

- When the AS is unavailable, the Node can still continue its operations.
  - It of course no longer can shop for Apps in the AS during the outage
  - It can still be fully managed by the MC, even Apps can be restored since the MC is independent from the AS

3: The Node is *highly independent* on the partners, but *rather dependent* on some.

- Although the Node can still continue all of its operations when a partner is unavailable, when an EnergyMarketParticipant is not available the participant cannot make allocations for energy flexibility offers that the node submits.

4: The partners, such as EnergyMarketParticipants, are *highly independent* from the MC

  - When the MC is unavailable, partners can still continue their communication with the account nodes, as long as their certificates did not expire.

5: The AS is *partially independent* from the MC

[20] For multi-location accounts, we have multiple Nodes. There the primary node is the most important one. Of course, when secondary nodes fail, the remote devices it monitors and controls are no longer accessible.

- When the MC is unavailable, the AS can still offer the shop experience for known accounts with verified certificates as long as the certificates do not expire.
    - It can even offer the purchase of Apps, however the MC is needed to support the actual download. Therefore, when a Node purchases an App while the MC is unavailable, it will have to download it later when the MC is available again.

6: The MC is *fully independent* from the AS

- When the AS is unavailable, the MC can still continue its operations
    - Although Apps are purchased in the AS, the MC manages the actual download and stores the App in its library. The MC therefore is independent from the AS to deal with restore/update actions that involve Apps, since it has all of the purchased Apps in its library.

3.2.2 *Nodes "highly independent" from the Management Center*

We examine the *high independence* of the Node from the availability of the MC in a bit more detail. When the MC is unavailable, the Node can still continue to perform all of its operational functions for its energy management.

- Certificate-based communications

As long as certificates are not expired, it can still

- Use protocols to communicate with its devices.
- Communicate to certified partners such as EnergyMarketParticipants.
- Offer interactions with its users via GUI and web services.

When a partner-certificate expires, communication with that partner stops since the partner cannot renew its certificate by the MC. When a new (just renewed or previously unseen certificate) is used for communication, the Node cannot communicate with that partner since it needs to receive certificate information from the MC to validate the certificate and authorize the communication.
It could be a policy to continue communication with expired certificates for some time, but this violates the security management.

- Management functions

When the MC is unavailable, the Node can still continue its local management functions, but

- It can no longer sync its data with the MC.

It keeps everything local until either the MC becomes available to resume the sync, or when the data volume outgrows its capacity. Depending on the policy it can either delete historic data and continue its operation, or halt operations.
So during the MC unavailability, there is a risk that data is not synced with the MC.
In case of Node failure in that period, data and the latest user settings could be lost, so that upon restore one starts with outdated user settings (the settings saved by the previous successful sync).

- Authorization based actions

The node can no longer perform operation for which it needs direct authorization by the MC. These operations are for example about adding new Users to the Account, adding new UserDevices to the Account, etc.

- Software updates, bundles and Apps

The node cannot download a previously bought APP, since the MC handles the download process. The node cannot receive system software updates, bundle updates (Plugins or Apps). This means that for example adding support for a new protocol is not possible during the MC unavailability.

3.2.3 *Nodes "highly independent" from the store availability*

The Nodes are *highly independent* from the availability of the AS.
When the AS is unavailable, users of the account loose the experience to shop in the store. During the unavailability period they cannot browse the store and purchase apps. When the AS would become unavailable during the download processing of an App purchase, there is no effect for the Node, since the download of the App is fully managed by the MC. The MC has copies of all purchased Apps, therefore the monitoring/repair/restore/update management functions of the MC for the Node can fully take place even when the AS is unavailable, since the MC is fully independent from the AS.

3.2.4 *Security and Privacy*

Related with Security and Privacy, the FP Application Infrastructure platform is designed to establish:

- **Secure** communication
- **Certified** communication between trusted parties
- **Authorization** to perform actions
- **Approval** of software components such as Apps
- **Verification** of software components such as Apps
- **Authentication** of software components such as Apps
- **Protection** against intrusion
- **Privacy** of sensitive user data
- **Rights Management** to perform actions

The different communications are illustrated in the next diagram.

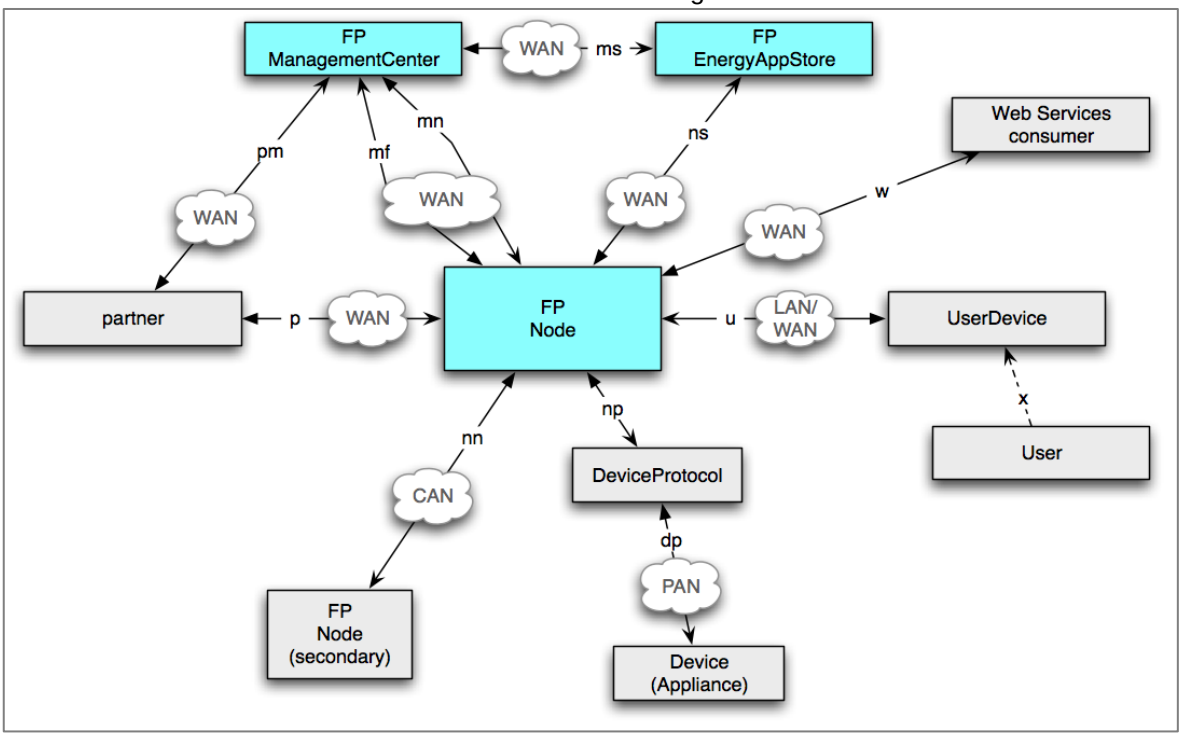


Figure 3-14 Communication and network types

The next table contains observations using references from the previous diagram.

Ref	Description

mn	The communication between management center and (primary) node is between two FP Net system components and happens over a WAN (internet). It can make use of the FP Application Infrastructure technical software framework services (OSGi).
mf	The communication between the management center and the node on low-level firmware level serves firmware installations and low-level management of the Node as an embedded system. It happens over a WAN (internet) and could use protocols such as TR-069.
ms	The communication between the management center and the store also happens over a WAN (internet). It is communication between two FP Application Infrastructure platform systems and could also make use of the FP Application Infrastructure technical software framework services.
pm	The communication between partners and the management center, for example to request/issue/renew/revoke partner-certificates, happens over a WAN (internet).
p	The communication between partners and the (primary) node also happens over a WAN (internet). The communication protocol should be secure but rather straightforward to pose a low-entry barrier for partners to participate in the FP Application Infrastructure platform.
np	The connection between the node (primary or secondary) and the protocols to monitor/control devices (DeviceProtocols), realized by a ProtocolConnector, links the node with a DeviceProtocol.
dp	Communication between a node and its devices uses a DeviceProtocol for nearby devices. DeviceProtocols, such as ZigBee, are typically protocols used on top of a Personal Area Network (PAN).
nn	Communication between a primary node and a secondary node happens over a Campus Area Network (CAN) or makes use of a WAN (internet).
u	The communication between UserDevices and the node, to access the GUI, typically happens over the Home Area Network (HAN / LAN), but can also use a WAN (internet).
x	When a User interacts with a node on his UserDevice, the communication session is also subject of security measures.
w	Consumers of the Web Services offered by a node communicate typically over the internet (WAN). These Web Services can of course also be consumed by User Devices on a LAN or WAN.

The methods used to communicate between entities should use secure communication protocols, using authentication and encryption.

The FP Application Infrastructure platform uses Certificates to certify parties. **The MC serves as certification authority.**

The FP Application Infrastructure platform uses explicit authorization for actions. The node has to ask explicit authorization to the MC for specific actions. **The MC serves as authorization center**. The Security Management Framework of the Node stores certificate and authorization information, so that it can independently operate until certificates expire or authorizations are revoked.

Software components, such as Apps, need approval and verification. **The AS is responsible for approval of Apps before offering them in its store for purchase.** The approval process should cover an inspection for compliance to the standards.

The FP ManagementCenter is responsible for security verification of Apps (and also for its own Plugins, software and firmware). This implies that, although an App, offered by the AS in its store, purchased by a User from a Node and installed (by the MC managing the download to the Node) still requires an authorization to be activated. Such authorization requires passing an obligatory security verification process at the MC. Therefore, the SMF of the Node has to ask authorization to the MC to use the App (or Apps of an AppGroup). The MC should only give authorization to Apps, which it has verified for security. The MC can also revoke authorizations when it discovers a security risk.

Authentication of Apps is realized by the use of digital signatures[21], so that the Security Management Framework of the FP Node can authenticate a provisioned App, to make sure that the integrity of the App received is verified.

The SMF of the Node is responsible to protect the Node against intrusions. It should prevent unauthorized access to the Node and the Account devices. It should also safeguard the node against an overload of incoming traffic from partners.

The FP Application Infrastructure platform should protect the privacy of sensitive account data. This implies that all communications should be encrypted, that all information stored in the MC should be safeguarded. It also implies that accounts on signup should be informed and give consent about the use of their user data and the data their node will send to the MC.[22] In case the MC offers data to other parties, it should use anonymization techniques and inform its accounts. The latter example could enable energy profile analysis or the development of analytic services (learning/forecasting) to offer information via an App as Third Party Information Provider.

The SMF of the Node is responsible for rights management, especially with respect to actions performed by third party components such as Apps.

3.2.5 Protocols

3.2.5.1 Protocols and security

The scope of the security management of the SMF of a NODE includes the security of the used DEVICEPROTOCOLS, but makes use of their security provisions. Therefore, the SMF has a firewall function to safeguard its PROTOCOLCONNECTORS, but on the other hand it delegates security responsibilities for the protocol network to the coordinator of that protocol network.

For example, using ZigBee, the PROTOCOLCONNECTOR communicates with the ZigBee coordinator (ZC), which acts as the trust center and repository of security keys of the ZigBee network.

3.2.5.2 Installation of support for new protocols

It is considered acceptable that the installation of a new protocol (and the installation of a new PROTOCOLCONNECTOR) requires a restart of the FP NODE software. It is not a design requirement that the installation of new protocol support has to be realized as a bundle update process not requiring a restart.

3.2.5.3 ProtocolConnectors and DeviceProtocols

The FP NODE contains PROTOCOLCONNECTORS that can be shared by multiple RESOURCES. The PROTOCOLDRIVERS (or their implicit parts in the RESOURCEDRIVERS) connect with the PROTOCOLCONNECTORS to communicate via the DEVICEPROTOCOL.

Although the complete functional decomposition will only be described in the chapter 4, the following diagram already shows a functional view of an FP Node to illustrate the relation with protocols.

---

[21] Reference implementation 12.11 uses JAR signing and verification. Apps are collections of modules which are jars/bundles. The Apps are only placed in the AppStore catalogue when their jars are signed. The provisioning process performs signature verification before installation.

[22] Note that the current design has FP Nodes syncing their data to the FP ManagementCenter. This design empowers rich possibilities with respect to analysis and recovery. It however needs further investigation if such scheme is considered feasible for all parties involved. If not, one could realize the syncing functionality in a similar way to local storage. This would preserve the functionality with respect to recovery, however it would limit severely the potential applications related with data analytics.

We show the abstraction of a Device as a Resource in the Resource Abstraction Layer of the FP Node. A Resource Manager App and a Resource Driver App represent the Resource. The ProtocolDriver[23] of the ResourceDriver talks to the ProtocolConnector for the used protocol. Note that the FP Node can support multiple DeviceProtocols with a ProtocolConnector for each protocol. Multiple devices can use the same protocol, the ProtocolDriver of their Resource communicates via the ProtocolConnector for that protocol.

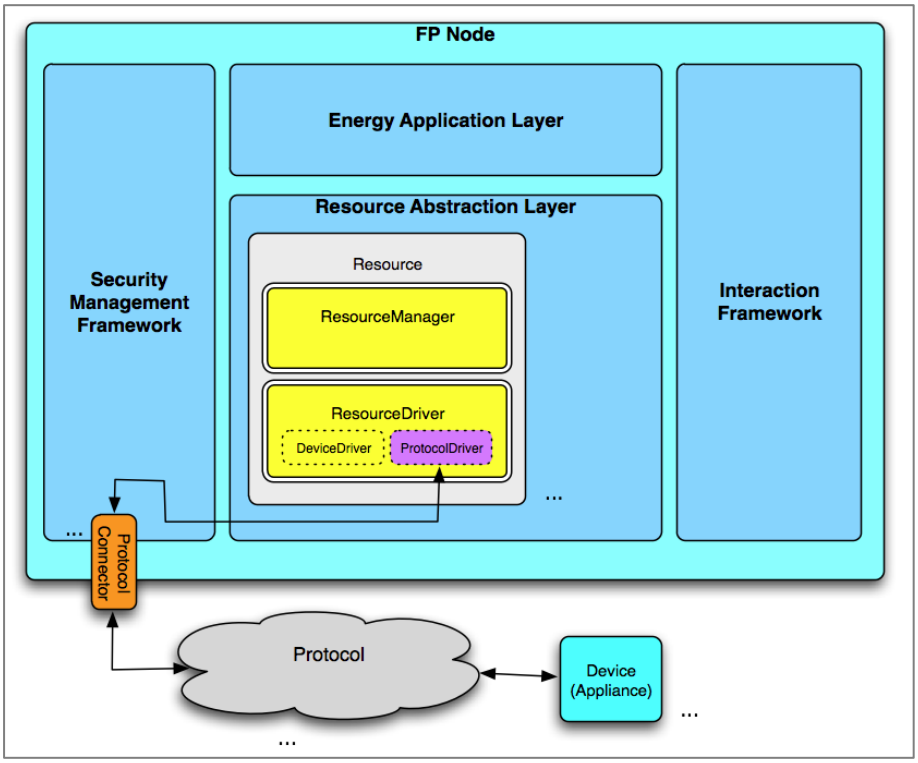


Figure 3-15 Protocol communication between Resource and Device

3.2.5.4 *Obtaining protocol support*

A ProtocolConnector is an optional component, but not necessarily an App, since its installation or update could require a restart and therefore it is not a bundle. If it is not an App, it is a component of the FP Node software provided by the MC. The choice to install a ProtocolConnector is typically an administrator task.

The information of the already supported protocols should be used to indicate/limit what Apps one can buy in the AS, or should be used to determine which ResourceDriver one should buy to use the App with the already supported protocols.

[23] A Resource Driver App has functionality specific for the device and related to the protocol. We present here these functions as DeviceDriver and ProtocolDriver components of a Resource Driver App, although these are functional components only which are not required to be separate technical components of the application.

3.3 Interactions

3.3.1 *Introduction*

In the following we introduce the main interactions between the 1st level entities of the FP Net. We describe the functional communication and communication rules.

In the next chapter, on the functional decomposition, we will give a formal overview of all communication interfaces. There we will present all communication functions, grouping them in external communication interfaces, providing details on who is required and who is optional in their provision, elaborating on relations with the component internals.

In the next paragraphs, we first introduce the main interactions and provide some explanation and motivation for the methods chosen.

3.3.1.1 *Functions*

We now describe the functions of the interfaces to realize the interactions.
These are functional descriptions that should receive an appropriate technical implementation.

```
A➔B.m()
```

This notation means that an entity of type A sends a message m to entity B, or alternatively calls a method m of the interface of entity B. Typically in a technical realization, B will represent a proxy for the entity B, so that an underlying message implementation can realize this using local/remote messaging.
The functional interface description assumes asynchronous communication; therefore we also include response messages in the interface descriptions.
Most interactions are secured using certificates; therefore most notations are of the following nature, where the certificate is supplied as one of the arguments.

```
A➔B.m(..,Certificate sc)
```

This notation specifies that certificate “sc” (the sender certificate) is the certificate of A that A supplies when sending the message m to B, so that B can authenticate A using its certificate sc. In the next paragraph, we describe the interactions involved in certification. We use the notation **Σ** to abbreviate this certificate argument; therefore the next notation is equivalent with the previous one:

```
A➔B.m(..,Σ)
```

3.3.1.2 *Rules*

We denote functional rules that the implementation must comply with using the following notation:

```
Rule: ...
```

3.3.1.3 *Parties involved in interactions*

We use the abbreviations of the next table to identify parties (entities) involved in the communication, either as sender or as receiver of communication requests.

Symbol	Description	Type of entity
mc	The management center	FP MANAGEMENTCENTER
n	A primary node	FP NODE (primary)
as	the store	FP APPSTORE
emp	An EnergyMarketParticipant	ENERGYMARKETPARTICIPANT
sup	A Supplier	SUPPLIER
d	The DSO	DSO
t	A Third Party Information Provider	THIRDPARTYINFORMATIONPROVIDER
p	One of: n,s,emp,sup,d,t	-

3.3.2 *Certification and authentication*

Parties in an FP Net use direct authentication to obtain certificates from the MC.

p➔mc.askCertificate(Credentials nc)

Note that this is an exception to the general rule that one should supply a sender certificate; in this case the sender p cannot supply its certificate since it is in the process of obtaining it.

The MC can issue a certificate with:

mc➔p.issueCertificate(Certificate issuedCertificate, Σ)

The MC also supplies its own certificate in communications.

The credentials for a node are typically the account identification and the node identification. The credentials for others are based on the relationship established when they subscribe a contract with the MCO to become a player in the FP Net. In this way all parties share a trust relationship and can use direct authentication to obtain certificates.

The certificate holder (party A) can supply the certificate with all of its communication as a token to prove its identity to another party (party B) it establishes communication with. Party B can then verify the certificate to authenticate B, typically without needing the MC to verify the certificate.

A➔B.sendMessage(MessageContent c, Σ)

This way the parties gain independence from the MC in their communication, while the MC, as certification center, can still enforce control by denying or revoking certificates.

Parties can ask for renewal of certificates, since certificates expire.

p➔mc.askCertificateRenewal(Certificate oldCertificate, Σ)

mc➔p.issueRenewal(Certificate renewedCertificate, Σ)

Parties that receive a communication request with an expired certificate should refuse the communication request. They are allowed to send a response of denial.

Parties that receive a communication request without a certificate or with an invalid certificate should deny the communication request and not respond to it.

The MC can refuse requests for certificates or renewals, by either ignoring the request or sending denial messages.

mc➔p.denyRequestCertification(Σ)

mc➔p.denyRequestCertificationRenewal(Σ)

The MC can revoke a certificate, so that it becomes invalid even before it expired. It can inform the certificate holder h:

mc➔h.revokeCertificate(Certificate c, Σ)

More importantly the MC should broadcast the revocation to other parties so that they no longer authenticate based on a revoked certificate. Typically the MC broadcast this to all its nodes n.

mc➔n.broadcastrevokedCertificate(Certificate revokedCertificate, Σ)

Parties, such as the node, are not obliged to involve the MC in the verification of certificates.

This because certificates serve as tokens and they can be verified without MC involvement. Since there is a trusted relation between the node and the MC, the node could also ask the MC for verification of each new certificate it receives. This could offload the verification work from the node. The node can ask to verify the certificate

n➔mc.askCertificateVerification(Certificate certificate, Σ)

and the mc can respond with

mc➔n.provideCertificateVerificationStatus(Certificate certificate,VerificationStatus s, Σ)

where the VerificationStatus identifies successful or failed verification.

However, whether the node makes use of this capability or not, nodes should realize the following:

A node should keep track of the verified certificates, so that it can continue communication with known partners when the MC is unavailable, at least until the certificates expire.

3.3.3 *Authorization*

The MC serves as authorization center for the Node.

Nodes should ask authorization for important actions. It should keep track of the obtained authorizations to become "highly independent" from the MC.

In this way, it can continue its operations with all authorized actions while the MC is unavailable.

Nodes should ask authorization to use a newly acquired app.

This is a hard requirement and therefore is not a policy option; it is an essential practice for the overall security management.

In this way, although only approved apps by the AS are offered in the AS, the security verification by the MC can be enforced. The MC should grant authorization if it concerns an App that received security verification by the MCO, it should deny authorization otherwise. When it discovers potential security risks, it can actively revoke authorizations by broadcasting a policy change that everyone should ask authorization again, before further use.

n➔mc.askAuthorizationAppUsage(AppGuid a, Σ)
mc➔n.grantAuthorizationAppUsage(AppGuid a, Σ)
mc➔n.denyAuthorizationAppUsage(AppGuid a, Σ)

An AppGuid is a unique identification of an App or an AppGroup, assigned by the AS (see further). It is important to gain authorization for an entire AppGroup, since the node needs the ability to activate the entire group in proper sequence.

The MC can perform its authorization process by also taking into account the software configuration of the Node, because it possesses all configuration information for all of its nodes. This is important since the configuration information at the time of App purchase can be different from the actual configuration, furthermore it is not required for the AS to only allow purchases of Apps that are guaranteed to work for a specific node configuration. This would be difficult to know anyway for the AS, since requiring the AS to know about all configuration details would make it more dependent on the AS. Furthermore the configuration information it would obtain from the MC could be outdated and different from the actual configuration of the Node at the time of activation of the App.

Therefore as a principle, whenever there is a configuration change in the Node, the Nodes are obliged to ask authorization again for all of their Apps. This improves reliability and is not intrusive since transparent for the Users.

3.3.4 *Policies*

The SMF of the Node performs its functions based on policies. It retains all policies so that it is highly independent from the MC. However, the MC can change policies and needs to notify the nodes in question so that they comply with the changes. A specific example of a policy change was already mentioned before, where changing the policy to force re-authorizations of apps is used by the MC in its security management.

The MC can notify nodes of a change in policy by

mc➔n.policyChange(Policies changedPolicies, Σ)

where “changedPolicies” contains the specifications of the changed policies and also contains an identification (PolicySetIdentifier), so that the nodes can efficiently respond with their compliance for that set. The nodes need to reply that they are in compliance with the policy changes, by:

```
n➔mc.policyCompliance(PolicySetIdentifier id, Σ)
```

3.3.5 *User Management*

The following user management operations need to be supported:

- Add user to account
- Remove user from account
- Add UserDevice to account
- Remove UserDevice from account
- Change User role to
- Grant user to UserDevice
- Revoke user from UserDevice

The FP Node GUI, provided via its INTERACTIONFRAMEWORK, can provide these operations.

The SecurityManagementFramework of the Node retains a representation of the account Users, Account Roles, Role Rights, User Roles, UserDevices, etc. The INTERACTIONFRAMEWORK can interact with the SMF to change the representation accordingly. When the SMF syncs its representations with the MC, also the MC has an updated representation.

There is no need to provide a user management interface for the MC; it receives an updates user management representation from the data that is synced from the SMF of a Node.

One could provide the user management operations in the WebServices of the Node, but this is optional.

The MC is in control with respect to

- Assigning the Roles of an Account, since these Roles are the maximum permissions for users of the account
- Determining the access rights contained in a Role
- Setting the AccountType of an account
- Setting the EnergyType of an account

Therefore, the MC can use these interactions with the Node:

```
mc➔n.changeAccountRoles(ChangeId id,AllowedRoles r, Σ)
mc➔n.changeRoleRights(ChangeId id,RoleSpecification r, Σ)
mc➔n.changeAccountType(ChangeId id,AccountType t, Σ)
mc➔n.changeEnergyType(ChangeId id,EnergyType e, Σ)
```

where the node can each time respond with reference to the used id of the requested change.

```
n➔mc.acceptedChange(ChangeId id, Σ)
```

Alternatively, the mc could supply these changes in one message with a structured object containing all requested changes.

3.3.6 *Sync*

The SECURITYMANAGEMENTFRAMEWORK (SMF) of a Node performs local management of the Node and thereby keeps track of all its representations, information flow, external and internal communication messages. The exact nature of all this information will be specified in the section on the functional decomposition of the FP Node.

The SMF periodically syncs its information with the FP ManagementCenter for monitoring, analysis and archiving. This is an essential property of the FP Application Infrastructure platform since it allows the MC to perform restore/repair/installation operations while preserving the state of the Node that was last synced to the MC.

The diagram on the right shows the information that the SMF incrementally syncs with the MC.

Knowing the previously successful sync point "from", it can request a sync of a new increment [from,to].

While the sync request is ongoing the information stack grows further of course.

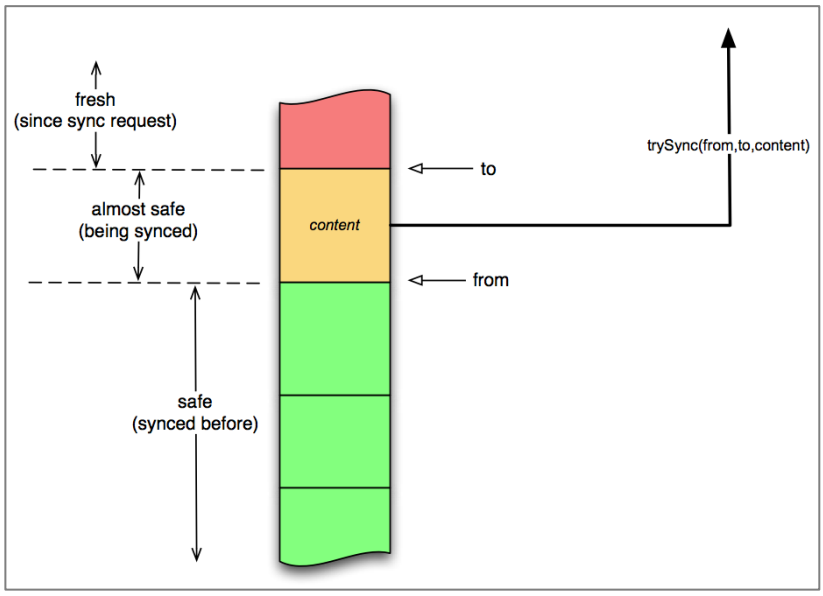


Figure 3-16 SMF syncing with the MC

The SMF of a Node can sync its latest increment by specifying the [from,to] interval as an IncrementIdentification and providing its content as an IncrementContent.:

```
n➔mc.trySync(IncrementIdentification id, IncrementContent c,Σ)
```

The MC can respond on a successful sync or on a failed sync:

```
mc➔n.succesfulSync(IncrementIdentification id,Σ)
mc➔n.failedSync(IncrementIdentification id,Σ)
```

If the mc does not provide a timely response, the SMF can also assume a failure.

Using these responses (or the lack of response), the SMF can determine the increment it needs to sync on its next sync request.
Note that all information since the last successful sync operation is potentially unsafe, since upon failure of the Node the information will be lost. It is dependent on a policy (provided by the MC) how frequent and on what type of actions or events a sync operation is required.
The internal communication from the INTERACTIONFRAMEWORK to the SMF of the Node needs to provide a "forceSync" operation, to give the GUI the ability to ensure a safe sync of the latest time-consuming settings that a user made.

3.3.7 *Monitoring / Restore*

To restore the Node after failure, the MC can ask to restore the information state. It can use the previously synced information from the node and force the restore of that information.

```
mc➔n.restore(IncrementIdentification id,IncrementContent c,Σ)
```

where the node can respond with

```
n➔mc.restoreSucessful(IncrementIdentification id,Σ)
```

This can be used for a full restore or an incremental restore to rollback. Therefore, on this request, the SMF makes sure that the node replaces its information for the specified increment ([from,to]) and removes information since that last increment endpoint (to), so that it comes in the correct state.

Further monitoring on a software level is not required since the MC can set the policy on regular sync operations and by inspecting this synced content it is able to perform high-level monitoring. There is room for additional low-level monitoring, using the firmware-level communication, typically using a protocol like TR-069.

The synced information not only contains the critical representations but also logging information of past events and messages. The MC can set and change the logging level to ensure that it receives logging information in the synced content at the right level of detail. Therefore the MC can set the logging level with:

mc➔n.setLoggingLevel(LoggingLevel lev,Σ)

where the node can respond with

n➔mc.loggingLevelSet(LoggingLevel lev,Σ)

3.3.8 *App purchases*

The purchase of an App in the AS makes use of the AS web services, which functionally must support the following:

n➔as.askPurchase(AppGuid g,Σ)

where the AS can respond with

as➔n.purchaseConfirmed(AppGuid g,Σ)

The provision/download of the App is managed by the MC.

When the AS approves a candidate App and offers it in its store for purchase, it should assign an AppGuid, which is an App identifier unique in the FP Net for that specific version of the App.
An AppGuid either identifies an individual App version, or identifies an AppGroup containing multiple Apps (which again have AppGuids).

Unique identification of Apps is important to efficiently communicate about Apps and ask/grant authorizations for their usage.

To adapt the view of the store or to filter the Apps in the store appropriate for the Node, the AS needs to take into account the Node configuration. For example the supported DeviceProtocols can be important to highlight or show only the AppGroups for ResourceManagers and ResourceDrivers compatible with already supported protocols.

Therefore, the node can submit some configuration details to the store. Alternatively, the AS could ask for configuration information to the MC.

n➔as.submitConfiguration(ConfigurationInformation c,Σ)

The final check on the real configuration of the node is performed when the node asks authorization to the MC to use the App.

3.3.9 *Store*

The AS can ask account information to the MC to perform its billing when Apps are purchased. It therefore supplies the certificate of the node that wants to make a purchase.

as➔mc.askAccountInfo(Certificate nodeCertificate,Σ)

The MC can respond with account information

mc➔as.supplyAccountInfo(AccountInformation info,Certificate nodeCertificate,Σ)

The AS needs to ask the MC to handle the download for a purchased App.

as➔mc.askHandleDownload(AppGuid g,Σ)

When the MC already has the necessary software of that App (or AppGroup), it can proceed. If not, it can as the AS for the software components.

mc➔as.askSoftware(AppGuid g,Σ)

and the AS can respond with

as➔mc.provideSoftware(AppGuid g,SoftwarePackages s,Σ)

When one wants to avoid the need for transactions, one could make sure that the purchase confirmation introduced earlier

```
as➔n.purchaseConfirmed(AppGuid g,Σ)
```

is only given when the AS already provided the software to the MC.
In that case the AS needs and additional message to ask if the MC already has the AppGuid software components

```
as➔mc.alreadyProvided(AppGuid g,Σ)
```

and based on the response

```
mc➔as.possessionindication(AppGuid g,Boolean b,Σ)
```

it can provide it with the "provideSoftware" message seen earlier.
All of this before returning the purchaseConfirmed message to the node.

A store probably wants App usage information to optimize the store experience with App ratings. One could introduce

```
as➔mc.requestAppUsageInformation(AppGuid g,Σ)
```

which can be answered by the MC with

```
mc➔as.supplyAppUsageInformation(AppGuid g,UsageInformation u,Σ)
```

3.3.10 *Software updates*

The interactions to provide updates of software components (software system, bundles, Plugins and Apps) are not specified here, since this will fully rely on the OSGi services.
This interface will be referred to as the **OpenServices** communication interface.

3.3.11 *Primary and Secondary Node interactions*

There are specific interactions between a primary node and secondary nodes for a multi-location account. These will be described further in the functional decomposition of the FP Node.

On a high level, all interactions with partners (EnergyMarketParticipants, Suppliers, DSO, ThirdPartyInformationProviders) and with the FP ManagementCenter and the FP AppStore are performed with the primary Node.

Only the firmware (TR-069) level interactions are also performed towards the secondary nodes.

This means that

(1) The SMF of the primary node needs to forward information to the SMF of each secondary node about:
    - Installing Plugins and Apps
        - Especially installing ResourceManagers, ResourceDrivers and ProtocolConnectors on secondary nodes
    - Policy changes, authentications to run apps, …

(2) The SMF of a secondary node needs to sync its data to the SMF of the primary node

(3) The SMF of a secondary node needs to forward the messages from the ResourceAbstractionLayer of the secondary node to the primary node, so that the EnergyApplicationLayer of the primary node can coordinate everything.

(4) The SMF of the primary node needs to forward the messages from the EnergyApplicationLayer that have resources of a secondary node as destination.

3.3.12 *Energy Management Interactions*

In general communication between partners and the node proceeds via a general mechanism already described higher, where based on certificates messages can be exchanged from A to B and vice versa.

```
A➔B.sendMessage(MessageContent c, Σ)
```

One could make the energy management communications with EnergyMarketParticipants more explicit by offering messages such as

```
n➔emp.submitProposal(EnergyProposal ep, Σ)
emc➔n.submitAssignment(EnergyAssignment ea, Σ)
```

where “ep” is an energy proposal (being a specific-format energy proposal in the protocol of the specific EnergyMarketParticipant) and where “ea” is an energy assignment (being a specific-format energy assignment that is a specific format from an EnergyMarketParticipant. Recall that an Energy Application knows these specifics of its EnergyMarketParticipant.

3.3.13 *Discovery and Handshaking*

Parties in an FP Net can communicate with each other. They authenticate with certificates obtained from the MC. However external parties such as EnergyMarketParticipants, Suppliers, DSO and ThirdPartyInformationProviders do not have information about what nodes they could talk to, furthermore for privacy and security purposes they should not have access to that information without the consent of the accounts. Account nodes initially also do not know what external parties they could communicate with.

We therefore need a communication flow to realize a discovery and handshaking process so that nodes can ask to the MC who they could talk to, so that they can initiate the communication with those parties via a handshaking process. By putting the initiative with the account nodes, we realize the privacy and security goal that external parties only know about account nodes that wish to communicate with them.

We already have introduced that everyone, also the external parties, need to ask a certificate to the MC. By this requirement the MC knows about everyone, also the external parties.

3.3.13.1 *Discovery*

A primary node can ask the MC about available parties with the following, where a DiscoveryRequest contains a DiscoveryRequestIdentifier and a DiscoveryCategory:

```
n➔mc.discover(DiscoveryRequest dr, Σ)
```

The DiscoveryCategory consists of a DiscoveryType and a DiscoverySpecification. Possible DiscoveryTypes are {EMP,SUP,DSO,TPIP,STORE,SEC}[24] to specify what type of party one wishes to discover. The DiscoverySpecification gives information about further requirements which type of party on wished to discover. This can be a DestinationIdentification such as an AppGuid. This is used for example to discover EnergyMarketParticipants that can work with a EnergyApplication app/plugin. The MC also knows about Apps and Plugins and can match these with the partner info is already collected from parties asking certificates. This can also be used in general to discover ThirdPartyInformationProviders that can provide information for an App. The DiscoverySpecification can also be an InformationFeed, used to discover parties that can provide information for Apps that consume/subscribe to an InformationFeed. For some combinations no DiscoverySpecification is needed since the MC has all the information it needs to resolve the request.

Representing a DiscoveryRequest with <DiscoveryRequestIdentifier, DiscoveryCategory> and a DiscoveryCategory with [DiscoveryType,DiscoverySpecification] we have the following possibilities:

```
n➔mc.discover(<#,[EMP,<EnergyApp AppGuid>]>, Σ)
n➔mc.discover(<#,[STORE,, Σ)
n➔mc.discover(<#,[SEC,, Σ)
n➔mc.discover(<#,[TPIP,<AppGuid>]>, Σ)
n➔mc.discover(<#,[TPIP,<InformationFeedType>]>, Σ)
```

[24] EMP = EnergyMarketParticipant, SUP = Supplier, DSO = DSO, TPIP = ThirdPartyInformationProvider, STORE = PM/AppStore, SEC = secondary node of the account . Primary nodes can ask to discover secondary nodes, not the other way around. Therefore we have a SEC DiscoveryType but not a PRIM type.

```
n➔mc.discover(<#,[DSO,>, Σ)
n➔mc.discover(<#,[SUP,]>, Σ)
```

The MC can respond to such a request with

```
mc➔n.discovered(DiscoveryRequestIdentifier dri,DiscoveredParties dp, Σ)
```

The DiscoveredParties object contains zero or more parties that the MC discovered that match the request. The MC returns all parties that have valid certificates and that match the request. Parties that did not yet announce themselves with the MC (by asking a certificate) are of course not included. A node can periodically request again to discover additional parties. It certainly requests discoveries upon installation of a new App/Plugin and after a restart/restore.

3.3.13.2 *Handshaking*

Having discovered a party x, the node performs a handshaking process with that party with

```
n➔x.hand(DiscoverySpecification ds,Σ)
```

The party x can respond with

```
x➔n.shake(DiscoverySpecification ds,Σ)
```

Note that the DiscoverySpecification is included, which again is empty for some types but for some is quite important. It could be that a node wants to talk to a TPIP about an AppGuid and not about its InformationFeed. This makes a difference, since in the first case the TPIP will start providing information for that App, in the latter case it will start providing information of the feed which the node will direct to all Apps subscribing to that feed.
Note that in the next chapter using the classification of external communication interfaces, the discovery process will be realized by a management interface, the handshaking process will be realized by different interfaces specific for the parties.[25]

3.3.14 *Other interactions*

3.3.14.1 *Time clock synchronization*

The MC provides a time-clock synchronization service to everyone in the FP Net. With the NTP, Network Time Protocol, the system of the MC can synchronize with a time-synchronization service like time.nist.gov. All other entities in the FP Net can point their NTP daemon towards the MC that serves as the network time source. From this, if Java uses the system clock on the host upon which it runs, all Java processes will run to the same clock. The communication interface providing this functionality is called the *TimeSynchronization*.

3.3.14.2 *Supplier invoices*

It would be preferable to receive a structured invoice from suppliers to match the pricing information for their time slots with the energy management events. This would allow for meaningful feedback to the users about the real impact of their energy management.
One should investigate the option to provide an EDI (Electronic data interchange) standard for supplier invoicing with the FP Application Infrastructure platform to promote its use, so that the FP Application Infrastructure platform can incorporate pricing information in its user feedback mechanism.

3.3.14.3 *Failure notifications*

Nodes could signal App failures to the MC using

```
n➔mc.reportAppFailure(AppGuid g,Σ)
```

This info could trigger an investigation by the MCO analyzing the synced logging data. Upon discovery of a problem, the MC could deny future App authorizations for all nodes or for a subset of

[25] In the next chapter one will see that the discovery is part of the MU interface, whereas the handshaking is part of the P interface for EnergyMarketParticipants, the S interface for suppliers, the D interface for DSO, the T interface for third party information providers, the AS interface for the store and the F interface for primary nodes that shake hands with a secondary node.

nodes based on their configuration. It could signal a policy change so that nodes are forced to ask authorization again, to make sure that all nodes involved stop using the App.

3.3.14.4 *Configuration Services*

Using the TR-069 protocol, the MC can execute system level configuration operations to manage its nodes. These operations are not described in detail in this functional specification document, but are part of the technical specification. On a high level, the following functions are supported:

- Service activation and reconfiguration
  - Initial configuration of the service as part of zero-touch of one-touch configuration process
  - Service reestablishment (ex. after device is factory-reset, exchanged)
- Remote Subscriber Support
  - Verification of the device status and functionality
  - Manual reconfiguration
- Firmware and Configuration Management
  - Firmware upgrade/downgrade
  - Configuration backup/restore
- Diagnostics and monitoring
  - Throughput (TR-143) and connectivity diagnostics
  - Parameter value retrieval
  - Log file retrieval

In the decomposition, we will refer to this communication interface as ConfigurationService.

3.3.14.5 *Information Collection requests*

Nodes retain information about their "state" and sync it regularly with the FP ManagementCenter (MC). They only have to keep a limited time window of information. When information is outdated or no longer needed for their current operation and when that information is already synced with the MC, nodes can clean up that information.

To make the practice of only keeping a limited window of information local at the Node transparent, information requests from the node logic components exceeding this window are forwarded to the MC. When for example a control logic component needs information to display a curve of historic energy consumption that stretches beyond the information window available at the node, this request is forwarded to the MC to perform the data collection.

Therefore a node can send an information collection request to the MC with

```
n➔mc.askInformationCollection(InformationCollectionRequest icr,Σ)
```

which the MC can respond to with

```
mc➔n.provideInformationCollection(InformationCollectionRequestIdentifier
id,InformationCollection c,Σ)
```

The InformationCollectionRequest 'icr' contains an InformationCollectionRequestIdentifier that is used in the response.
Secondary nodes use this to forward information collection requests to their primary node.[26]

[26] In the decomposition of the next chapter, one will find an extensive propagation example of these information collection requests, from secondary node over primary node, even over different levels of information in the management center.

4 FP Application Infrastructure Functional Decomposition

4.1 Main components and external interfaces

4.1.1 Main entity components

Main entity components of an FP Net	*Number*
FP ManagementCenter	Single for all accounts in an FP Net.
FP AppStore	Single for all accounts in an FP Net.
FP Node	For each account: 1 primary, optional multiple secondary ones
EnergyMarketParticipant	There can be multiple in an FP Net, there can be multiple interacting with an account.
Supplier	There can be multiple in an FP Net, there can be multiple interacting with an account.
DSO	1 (since an FP Net corresponds to a Distribution Net)
ThirdPartyInformationProvider	There can be multiple

4.1.2 Notations of external communication interfaces

We use the following notations to identify the external communication interfaces of the main entity components of the FP Application Infrastructure platform. The interfaces are shown with the notation of the next figure:

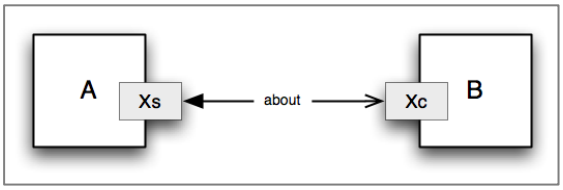


Figure 4-1 External communication interfaces

When two main entity components A and B communicate about a subject X, the component A provides a communication interface Xs and component B provides a communication interface Xc. The interface Xs is called the server side interface of X; the interface Xc is called the client side interface of X. The communication is asynchronous; the choice of terminology for being server of client is rather intuitive and does not have formal implications. An arrow shows the communication path; the solid arrowhead indicates the direction of the communication where the most initiative is taken, again an intuitive meaning without formal implications. There can be optional text on the arrow describing the overall subject of the communication.

The arrows in a diagram can be connected, as in the previous figure. When this would overload a diagram they can also be shown without connections. In that case we show the correspondence between both ends by using a common symbol. The next figure shows an example of this where the common square symbol is used to indicate the correspondence.

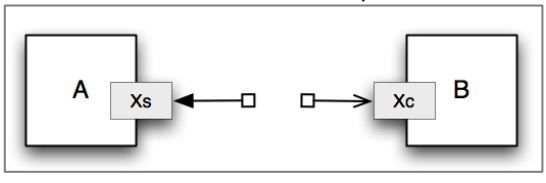


Figure 4-2 Unconnected external communication interfaces

4.1.3 *Overview*

The next diagram shows the main entity components with their external communication interfaces. The interfaces are described in detail in the following paragraph.

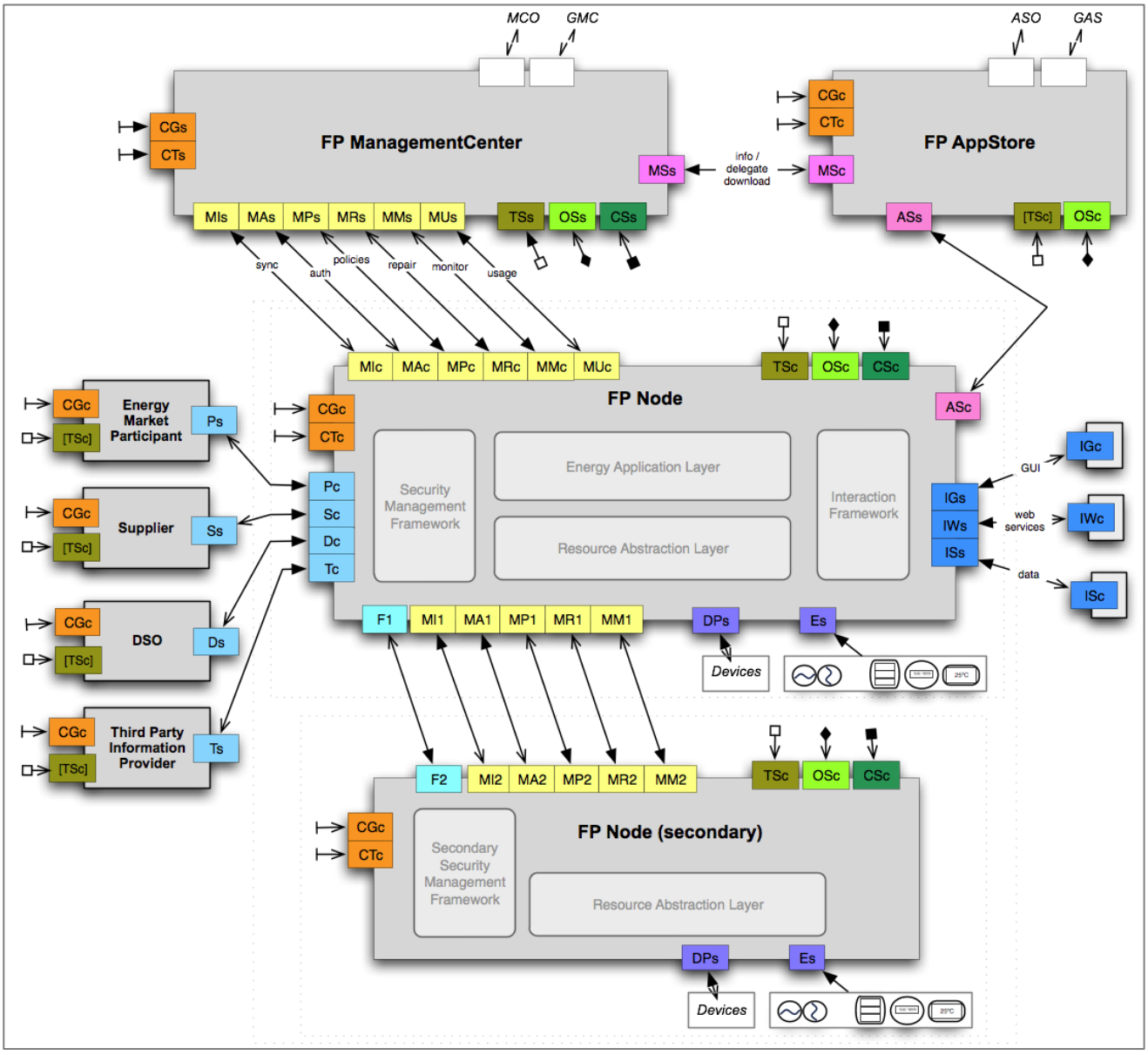
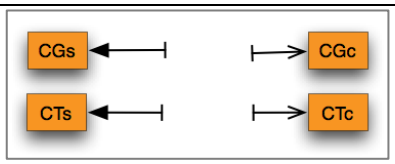


Figure 4-3 Main components and external communication interfaces

4.1.4 External Communication Interfaces

4.1.4.1 Communication Interfaces about Certification



The communication interfaces about certification provide all entities in the FP Net a secure way of communication by the use of certificates. The FP ManagementCenter serves as the certification centers. There are specific services for trusted partners.

4.1.4.1.1 CG - CertificationGeneral

Communication Interface		
CertificationGeneral	*Provides general certification services*	
interface component	*abbrev*	*provided by*
CertificationGeneralServer	**CGs**	FP ManagementCenter
CertificationGeneralClient	**CGc**	FP Node: all (primary and secondary) FP AppStore, DSO, Supplier, EnergyMarketParticipant, ThirdPartyInformationProvider

Specifications:	
CGs	
1	***askCertificate(Credentials nc)***
4	***askCertificateRenewal(Certificate oldCertificate, Σ)***
CGc	
2	***issueCertificate(Certificate issuedCertificate, Σ)***
5	***issueRenewal(Certificate renewedCertificate, Σ)***
3	***denyRequestCertification(Σ)***
6	***denyRequestCertificationRenewal(Σ)***
7	***revokeCertificate(Certificate c, Σ)***

Parties ask a certificate with (1), receive a certificate with (2) or a denial with (3).
Certificates expire, one can ask for a renewal with (4), receive a renewed certificate with (5) or a denial with (6). The MC can revoke certificates with (7).

Parties should include a valid certificate with all of their communication.

Parties that receive a communication request with an expired certificate should deny the communication request. They are allowed to send a response of denial.

Parties that receive a communication request without a certificate or with an invalid certificate should deny the communication request and not respond to it.

4.1.4.1.2 CT - CertificationTrusted

Communication Interface		
CertificationTrusted	*Provides special certification services for trusted partners*	
interface component	*abbrev*	*provided by*
CertificationTrustedServer	**CTs**	FP ManagementCenter
CertificationTrustedClient	**CTc**	FP Node: all (primary and secondary) FP AppStore

Specifications:	
CTs	
1	***askCertificateVerification(Certificate certificate, Σ)***
CTc	

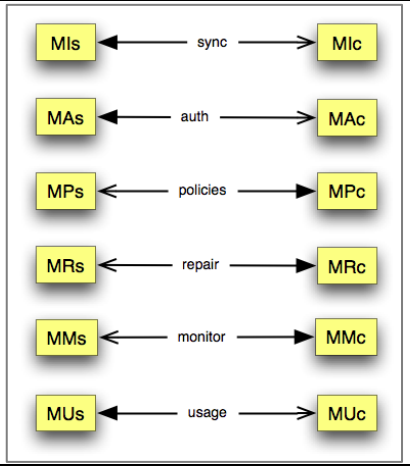
3	***broadcastrevokedCertificate(Certificate revokedCertificate, Σ)***
2	***provideCertificateVerificationStatus(Certificate certificate,VerificationStatus s, Σ)***

Since there is a trusted relation between the node and the MC, the node could also ask the MC for verification of each new certificate it receives. This could offload the verification work from the node. The node can ask the MC to verify the certificate with (1) and receive (2).
However, if the node makes use of this capability or not, in any case, nodes should realize the following:

A node should keep track of the verified certificates, so that it can continue communication with known partners when the MC is unavailable, at least until the certificates expire.

With (3) the MC can broadcast the revocation of a certificate to its trusted partners, so that they no longer authenticate based on a revoked certificate.

4.1.4.2 *Communication Interfaces about Management*



The communication interfaces about management are about the MC managing the Nodes. These are for primary nodes only. The interfaces deal with information sync services, authorization, policy management, repair services, monitoring services and communication of usage (users, roles, rights, ..) changes.
For most of them there are similar interfaces to communicate between primary and secondary nodes.
Lower level management tasks are supported by other interfaces (OS, CS, TS; see further).

4.1.4.2.1 *MA – ManagementAuthorization*

Communication Interface		
ManagementAuthorization	*Provides communication interface for primary nodes to ask authorization to the MC for actions. For secondary nodes there are similar interfaces MA1 and MA2 (see further).*	
interface component	*abbrev*	*provided by*
ManagementAuthorizationServer	**MAs**	FP ManagementCenter
ManagementAuthorizationClient	**MAc**	FP Node: only primary
Specifications:		
MAs		
1	***askAuthorizationAppUsage(AppGuid a, Σ)***	
MAc		
2	***grantAuthorizationAppUsage(AppGuid a, Σ)***	
3	***denyAuthorizationAppUsage(AppGuid a, Σ)***	

Nodes should ask authorization for important actions. It should keep track of the authorizations so that it is becomes highly independent from the MC.

Using an App is an important action that should be authorized.

With (1) the node asks authorization to use an App, with (2) it receives authorization, with (3) it receives refusal.

4.1.4.2.2 MP – ManagementPolicy

Communication Interface		
ManagementPolicy	*Provides communication interface to communicate policy issues. This is for the MC and the primary nodes. For secondary nodes there are similar interfaces MP1 and MP2 (see further).*	
interface component	*abbrev*	*provided by*
ManagementPolicyServer	**MPs**	FP ManagementCenter
ManagementPolicyClient	**MPc**	FP Node: only primary

Specifications:	
MPs	
1	***policyCompliance(PolicySetIdentifier id, Σ)***
MPc	
2	***policyChange(Policies changedPolicies, Σ)***

The MC notifies nodes of a policy change with (2), they respond with (1) when they are in compliance.

On policy changes, nodes must comply and respond their compliance, If they do not respond their compliance within a given time period, the MC considers them non-compliant and can take further actions (using its configuration interface CG, see further).

4.1.4.2.3 MU – ManagementUsage

Communication Interface		
ManagementUsage	*Provides communication interface for communicating usage (users, roles, rights) to the MC. It also contains the discovery interface.*	
interface component	*abbrev*	*provided by*
ManagementUsageServer	**MUs**	FP ManagementCenter
ManagementUsageClient	**MUc**	FP Node: only primary

Specifications:	
MUs	
5	***acceptedChange(ChangeId id, Σ)***
6	***discover(DiscoveryRequest dr, Σ)***
MUc	
1	***changeAccountRoles(ChangeId id,AllowedRoles r, Σ)***
2	***changeRoleRights(ChangeId id,RoleSpecification r, Σ)***
3	***changeAccountType(ChangeId id,AccountType t, Σ)***
4	***changeEnergyType(ChangeId id,EnergyType e, Σ)***
7	***discovered(DiscoveryRequestIdentifier dri,DiscoveredParties dp, Σ)***

Nodes (via a user with administration rights using its GUI) can make change concerning usage (adding users, user devices, changing roles, etc.). These changes require no MC interaction; they get communicated to the MC via the regular sync operations anyway (MI interface, see further).

The MC can change some aspects about usage itself and communicates this via the MU interface, such as (1) to communicate a change in the allowed roles for the account, (2) a change in the rights of a role, (3) a change of the AccountType, (4) a change of the EnergyType. The node acknowledges these changes with (5).

A node has to acknowledge a change in usage with (5). An acknowledgement means that the SMF of the node adapted its local representations (of users, user devices, roles, rights, etc.) and that the SMF is operating in compliance with these settings.

This communication interface also realizes the *discovery* process, discussed in the previous chapter. With (6) a primary node can ask to discover parties, with (7) the MC responds with the

parties discovered. Note that the *handshaking* process is realized by interfaces specific to parties (P,S,D,T,AS,F).

4.1.4.2.4 MI – ManagementInformation

Communication Interface		
ManagementInformation	*Provides management information communication for syncing data between primary node and management center. For secondary nodes there are similar interfaces MI1 and MI2 (see further)*	
interface component	*abbrev*	*provided by*
ManagementInformationServer	**MIs**	FP ManagementCenter
ManagementInformationClient	**MIc**	FP Node: only primary

Specifications:	
MIs	
1	***trySync(IncrementIdentification id, IncrementContent c,Σ)***
5	***loggingLevelSet(LoggingLevel lev,Σ)***
6	***askInformationCollection(InformationCollectionRequest icr,Σ)***
MIc	
2	***succesfulSync(IncrementIdentification id,Σ)***
3	***failedSync(IncrementIdentification id,Σ)***
4	***setLoggingLevel(LoggingLevel lev,Σ)***
7	***provideInformationCollection(InformationCollectionRequestIdentifier id,InformationCollection c,Σ)***

Primary nodes periodically sync their information to the MC, using increments. They submit an increment with (1), receive a response of a successful sync with (2) or failure with (3). A LoggingLevel determines the level of detail for logging the information. This can be changed by the MC with (4), which the node confirms with (5).

Nodes have to confirm a change in LoggingLevel. A change in LoggingLevel does not need imply that the next increment will be of that LoggingLevel, since the buildup of information of that next increment could already date from before the change request.

Information collection requests can be asked by (6) and responded to by (7).

4.1.4.2.5 MM – ManagementMonitoring

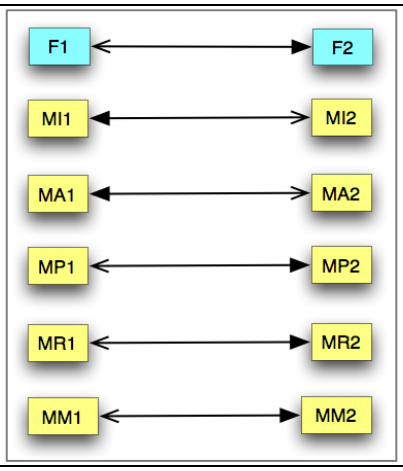
Communication Interface		
ManagementMonitoring	*Provides communication interface for monitoring of nodes by the MC. This is about FP Application Infrastructure software level monitoring, there is lower-level (firmware/configuration) monitoring defined by other CS interfaces.* *For secondary nodes there are similar interfaces MM1 and MM2 (see further).*	
interface component	*abbrev*	*provided by*
ManagementMonitoringServer	**MMs**	FP ManagementCenter
ManagementMonitoringClient	**MMc**	FP Node: only primary

Specifications:	
MMs	
	reportAppFailure(AppGuid g,Σ)
MMc	
	To be defined, depending on the technical choices (TR-069) for the low-level (device/firmware) configuration services in the CS interface.

4.1.4.2.6 MR – ManagementRepair

Communication Interface		
ManagementRepair	*Provides communication interface for management repair actions of the MC for nodes.* *For secondary nodes there are similar interfaces MR1 and MR2 (see further).*	
interface component	*abbrev*	*provided by*
ManagementRepairServer	**MRs**	FP ManagementCenter
ManagementRepairClient	**MRc**	FP Node: only primary
Specifications:		
MRs		
	`restoreSucessful(IncrementIdentification id,Σ)`	
MRc		
	`restore(IncrementIdentification id,IncrementContent c,Σ)`	
	Others to be defined, depending on the technical choices (TR-069) for the low-level (device/firmware) configuration services in the CS interface.	

4.1.4.3 Communication interfaces about management with primary & secondary nodes



These are communication interfaces to be used between primary and secondary FP Nodes. There is a general forwarding interface Fn. There are forwarding versions of the management interfaces so that the secondary node can interact with

4.1.4.3.1 MAn – ManagementAuthorization-Forwarding

Communication Interface		
MAforwarding	*Provides communication interface for primary-secondary node communication about authorization.*	
interface component	*abbrev*	*provided by*
MAForwardingServer	**MA1**	Primary FP Node when there are secondary FP Nodes for the Account.
MAForwardingClient	**MA2**	Every secondary FP Node of the Account.
Specifications: *The communication interface is the same as the equivalent management interface used between primary nodes and the management center. The interface specification is equivalent, however the server side is provided by the SMF of the primary node. The SMF also forwards management communication from the MC to the secondary nodes this way, so that the MC does not have to interact with secondary nodes directly for these management services.*		
MA1	Same as the equivalent MAs interface definition.	
MA2	Same as the equivalent MAc interface definition.	

4.1.4.3.2 MP – ManagementPolicy-Forwarding

Communication Interface		
MPforwarding	*Provides communication interface for primary-secondary node communication about policy management.*	
interface component	*abbrev*	*provided by*
MIforwardingServer	**MP1**	Primary FP Node when there are secondary FP Nodes for the Account.
MIforwardingClient	**MP2**	Every secondary FP Node of the Account.
Specifications: *The communication interface is the same as the equivalent management interface used between primary nodes and the management center. The interface specification is equivalent, however the server side is provided by the SMF of the primary node. The SMF also forwards management communication from the MC to the secondary nodes this way, so that the MC does not have to interact with secondary nodes directly for these management services.*		
MP1	Same as the equivalent MPs interface definition.	
MP2	Same as the equivalent MPc interface definition.	

4.1.4.3.3 MIn – ManagementInformation-Forwarding

Communication Interface		
MIforwarding	*Provides communication interface for primary-secondary node communication about information management.*	
interface component	*abbrev*	*provided by*
MIforwardingServer	**MI1**	Primary FP Node when there are secondary FP Nodes for the Account.
MIforwardingClient	**MI2**	Every secondary FP Node of the Account.
Specifications: *The communication interface is the same as the equivalent management interface used between primary nodes and the management center. The interface specification is equivalent, however the server side is provided by the SMF of the primary node. The SMF also forwards management communication from the MC to the secondary nodes this way, so that the MC does not have to interact with secondary nodes directly for these management services.*		
MI1	Same as the equivalent MIs interface definition.	
MI2	Same as the equivalent MIc interface definition.	

4.1.4.3.4 MM – ManagementMonitoring-Forwarding

Communication Interface		
MM-forwarding	*Provides communication interface for primary-secondary node communication about authorization.*	
interface component	*abbrev*	*provided by*
MMforwardingServer	**MM1**	Primary FP Node when there are secondary FP Nodes for the Account.
MMforwardingClient	**MM2**	Every secondary FP Node of the Account.
Specifications: *The communication interface is the same as the equivalent management interface used between primary nodes and the management center. The interface specification is equivalent, however the server side is provided by the SMF of the primary node. The SMF also forwards management communication from the MC to the secondary nodes this way, so that the MC does not have to interact with secondary nodes directly for these management services.*		
MM1	Same as the equivalent MMs interface definition.	
MM2	Same as the equivalent MMc interface definition.	

4.1.4.3.5 MR – ManagementRepair-Forwarding

Communication Interface		
MRforwarding	*Provides communication interface for primary-secondary node communication about repair management.*	
interface component	*Abbrev*	*provided by*
MRforwardingServer	**MR1**	Primary FP Node when there are secondary FP Nodes for the Account.
MRforwardingClient	**MR2**	Every secondary FP Node of the Account.
Specifications: The communication interface is the same as the equivalent management interface used between primary nodes and the management center. The interface specification is equivalent, however the server side is provided by the SMF of the primary node. The SMF also forwards management communication from the MC to the secondary nodes this way, so that the MC does not have to interact with secondary nodes directly for these management services.		
MR1	Same as the equivalent MRs interface definition.	
MR2	Same as the equivalent MRc interface definition.	

4.1.4.3.6 F - ForwardingServices

Communication Interface		
ForwardingServices	*Provides communication interface for primary-secondary node communication to forward secondary node energy information to the primary node. Also used for communication with embedded resources.*[27]	
interface component	*Abbrev*	*provided by*
ForwardingServicesServer	**F1**	Primary FP Node when there are secondary FP Nodes for the Account.
ForwardingServicesClient	**F2**	Every secondary FP Node of the Account (or embedded resource).
Specifications:		
F1		
1	***submitControlSpaceFromRemoteResource(ControlSpacecs,ResourceManagerId rm,Σ)***	
4	***shake(DiscoverySpecification ds,Σ)***	
	additional more fine-grained messages to be specified in the incremental development and design process	
F2		
2	***submitAllocationToRemoteResource(Allocation a,ResourceManagerId rm,Σ)***	
3	***hand(DiscoverySpecification ds,Σ)***	
	Additional more fine-grained messages to be specified in the incremental development and design process	

The handshaking process is realized by: the primary node using (3) and the secondary node responding with (4). An administrator user activating secondary nodes using the GUI control interface typically initiates this.

A secondary node of an account, monitors and controls devices in a secondary location. The SMF of the secondary node forwards information to the primary node, so that the primary node Energy Application Layer has a complete view on all Resources. Therefore, the Resources of the secondary Node communicate with the Energy Application Layer of the primary node. When they

[27] In the explanation we first focus on primary/secondary FP Node communication. However this interface is also used to work with embedded resources, where the resource manager and resource driver is embedded on the appliance. This situation is described in the following paragraph as well.

submit a ControlSpace, their SMF forwards this to the primary node using (1) where the “rm” is the Guid of the ResourceManager submitting the ControlSpace. When an Energy Application of the primary Node sends an allocation to a Resource, the SMF does not route it to a local Resource but sends the allocation to the secondary Node. Its SMF receives the allocation by (2) and routes it to the ResourceManager of its local Resource.

The figure on the right illustrates the forwarding practice between a primary Node and secondary Nodes.

It shows two Resources in the primary FP Node, three Resources is the 1st secondary FP Node and two Resources in the 2nd secondary FP Node.

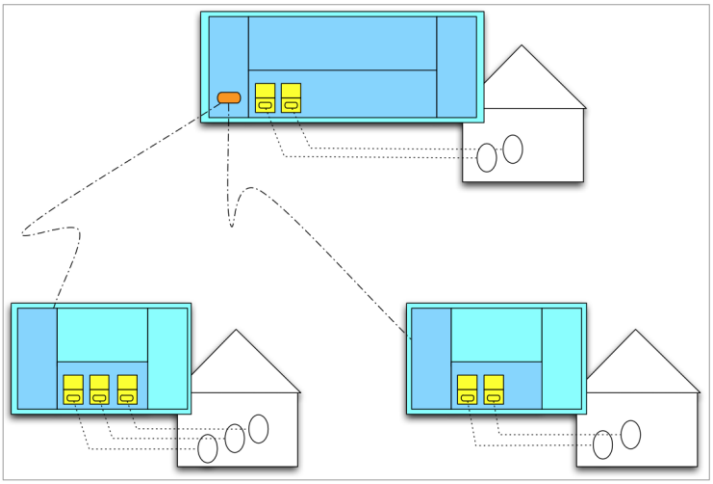


Figure 4-4 Forwarding between primary and secondary FP Nodes

For an Energy Application this practice is fully transparent, since the SMF handles the forwarding. From the viewpoint of an Energy Application, the situation looks as if the Node controls all Resources, local and remote ones. This viewpoint is shown in the figure on the right, where the Energy Application Layer provides a view on all resources, 2 local and 5 remote ones.

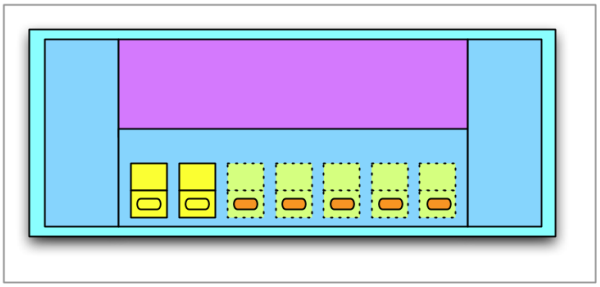


Figure 4-5 Energy Application Layer view on local and remote resources

The forwarding interfaces are not only used between primary and secondary nodes. They are also used to handle “embedded resources”. To illustrate that principle, the next paragraphs show three situations, first the regular situation without forwarding, then the primary/secondary node forwarding, and finally the forwarding for embedded resources.

4.1.4.3.6.1 Regular situation without forwarding

In the regular situation where devices are in close proximity (for nearby device protocols) they are abstracted in resources in the Resource Abstraction Layer on the (primary) FP Node. The forwarding interfaces are not applied. This situation is shown in the diagram on the right. The blue arrow shows the exchange of Control Spaces and Allocations between an Energy Application and the Resource.

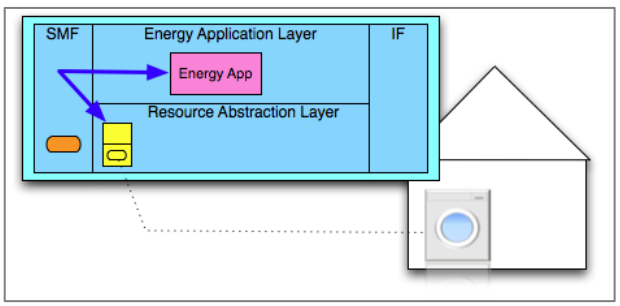


Figure 4-6 Resource on primary FP Node

4.1.4.3.6.2 *Forwarding between primary and secondary nodes*

For the situation where devices are not in close proximity, they are abstracted in resources on a secondary FP Node. The forwarding interfaces are applied for communication between the primary and secondary node. The diagram below illustrates this situation; again the blue arrow shows the exchange of Control Spaces and Allocation between an Energy Application and the now remote Resource.

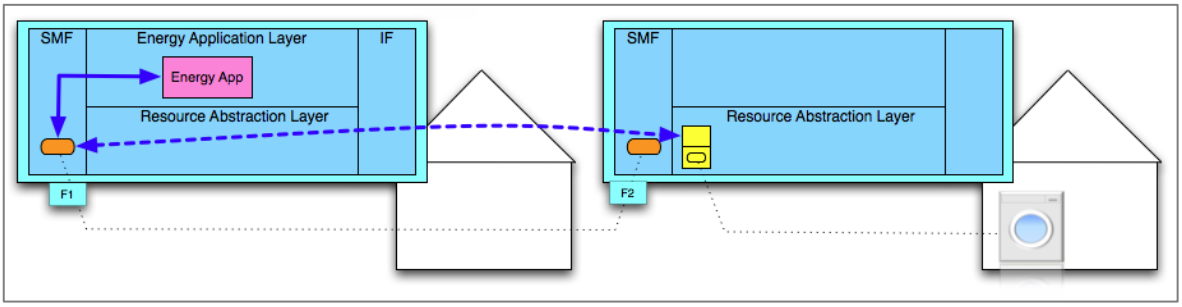


Figure 4-7 Resource on secondary FP Node

4.1.4.3.6.3 *Forwarding for embedded resources*

Besides the practice with primary and secondary nodes, the same (functional) forwarding interface is used for the situation of “embedded resources” (although it has a specific technical realization). With “embedded resources” we mean the integration of the resource manager and resource driver functionality embedded on the appliance itself. Again the forwarding interface is used in this situation. The diagram below illustrates this, for an embedded resource with a primary node.

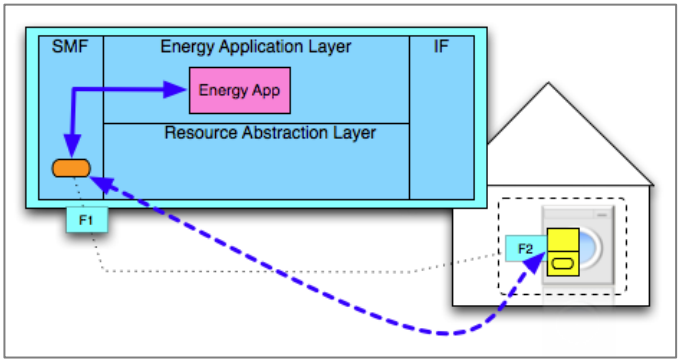


Figure 4-8 Embedded Resource with primary FP Node

The diagram below shows an embedded resource with a secondary node. The Energy Application of the primary node communicates via forwarding services with the secondary node, which again uses the forwarding interface to communicate with the embedded resource.

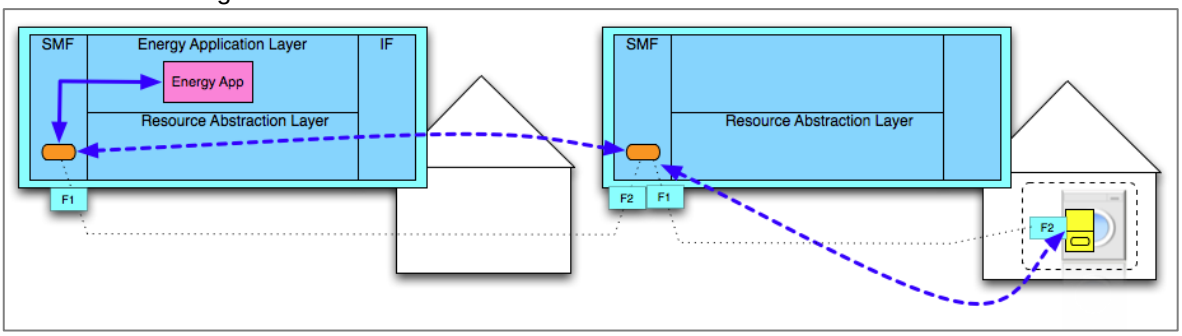
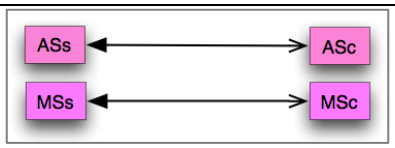


Figure 4-9 Embedded Resource with secondary FP Node

4.1.4.4 Communication Interfaces about Store



The AS is a communication interface between clients (Nodes) and the store (FP AppStore). There is also a communication interface between the management center and the store to exchange information.

4.1.4.4.1 AS – AppStore

Communication Interface		
AppStore	*Provides communication interface for nodes to shop in the FP AppStore. Note that this interface is typically used embedded in a GUI implementation for shopping in the store.*	
interface component	*abbrev*	*provided by*
AppStore Server	**ASs**	FP AppStore
AppStore Client	**ASc**	FP Node: only primary
Specifications:		
ASs		
3	***askPurchase(AppGuid g,Σ)***	
1	***submitConfiguration(ConfigurationInformation c,Σ)***	
5	***hand(DiscoverySpecification ds,Σ)***	
ASc		
4	***purchaseConfirmed(AppGuid g,Σ)***	
2	***configurationSubmitted(Σ)***	
6	***shake(DiscoverySpecification ds,Σ)***	

With (1) a node can submit (some of its) configuration details to the store. This is useful to adapt the view in the store compatible with the configuration (for example show only or highlight Apps or AppGroups for the already supported device protocols). Note that this configuration information is informative only. The final check on the real configuration of the node is performed when the node asks authorization to the MC to use the App after is has been purchased and downloaded.
With (3) a (user of a) node can purchase an App or AppGroup. When receiving the confirmation (4), this means that the AS has finished the purchase transaction, delegated the download to the MC and is sure that the MC is able to handle the download independently.[28]

```
When the AS approves a candidate App and offers it in its store for purchase,
it should assign an AppGuid, which is an App identifier unique in the FP Net
for that specific version of the App.
An AppGuid either identifies an individual App version, or identifies an
AppGroup containing multiple Apps (which again have AppGuids).
```

Handshaking is realized by the primary node using (5) and the store responding with (6).

4.1.4.4.2 MS – ManagementStore

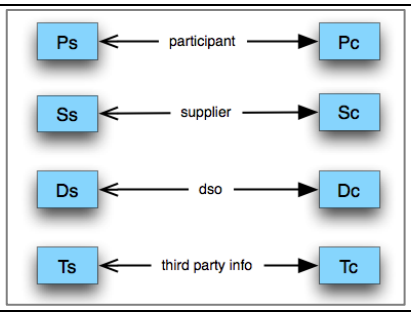
Communication Interface		
ManagementStore	*Provides communication interface for the FP ManagementCenter and the FP AppStore to exchange information.*	
interface component	*abbrev*	*provided by*
ManagementStoreServer	**MSs**	FP ManagementCenter
ManagementStoreClient	**MSc**	FP AppStore
Specifications:		
MSs		
1	***askAccountInfo(Certificate nodeCertificate,Σ)***	
7	***askHandleDownload(AppGuid g,Σ)***	

[28] To delegate the download process to the MC, there are provisions in the MS interface.

5	***provideSoftware(AppGuid g,SoftwarePackages s,Σ)***
3	***alreadyProvided(AppGuid g,Σ)***
	requestAppUsageInformation(AppGuid g,Σ)
MSc	
2	***supplyAccountInfo(AccountInformation info,Certificate nodeCertificate,Σ)***
6	***receivedSoftware(AppGuid g,Σ)***
4	***possessionindication(AppGuid g,Boolean b,Σ)***
	supplyAppUsageInformation(AppGuid g,UsageInformation u,Σ)

The MS interface is about communication between the store and the MC. With (1)-(2) the store can ask account info to handle its invoicing. With (7) the store can delegate the download of a purchase to the MC. However, before doing this, it should make sure that the MC has the software available for the App (or all Apps in the AppGroup). When confirming a purchase to the node, it should ensure that the MC is able to handle the download even if the store becomes unavailable. With (3) the store can ask if the MC already has the software, the MC responds with (4). The store can provide the software by (5) and get conformation by (6). Only then it should delegate the download by (7) and inform the node with a 'purchaseConfirmed' from the AS interface.

4.1.4.5 *Communication Interfaces about Partners*



There are communication interfaces between the node and partners in the FP Net, such as Participants (EnergyMarketParticipants), Supplier(s), DSO and ThirdPartyInformationProviders.

4.1.4.5.1 *P – ParticipantCommunication*

Communication Interface		
ParticipantCommunication	*Provides communication interfact for EnergyMarketParticipants to talk to FP Nodes.*	
interface component	*abbrev*	*provided by*
ParticipantCommunicationServer	**Ps**	EnergyMarketParticipant
ParticipantCommunicationClient	**Pc**	FP Node: only primary
Specifications:		
Ps		
1	***submitOffer(EnergyProposal ep, Σ)***	
3	***hand(DiscoverySpecification ds,Σ)***	
Pc		
2	***submitAssignment(EnergyAssignment ea, Σ)***	
4	***shake(DiscoverySpecification ds,Σ)***	

An EnergyProposal is a proposal for a specific EnergyMarketParticipant, which responds with an assignment to match the proposal; an assignment can also be a rejection.
Note that this is a functional communication interface only. The exact technical interface can involve more detailed exchange of messages. Furthermore, some EnergyMarketParticipants could have the ability to not only respond to offers but also to withdrawal of an offer.
Handshaking is realized by: the primary node using (3), with the participant responding by (4).

4.1.4.5.2 S – SupplierCommunication

Communication Interface		
SupplierCommunication		
interface component	*abbrev*	*provided by*
SupplierCommunicationServer	**Ss**	Supplier(s)
SupplierCommunicationClient	**Sc**	FP Node: only primary
Specifications:		
Ss		
1	***hand(DiscoverySpecification ds,Σ)***	
	To be determined ***received(SupplierInfoId id, Σ)***	
Sc		
2	***shake(DiscoverySpecification ds,Σ)***	
	To be determined ***sendSupplierInfo(SupplierInfo s, Σ)***	

There is optional communication possible between the Supplier(s) and the node. This could involve the exchange of a structured supplier invoice with pricing information listed by time period.
Handshaking is realized by: the primary node using (1) and the supplier responding with (2).

4.1.4.5.3 D – DsoCommunication

Communication Interface		
DsoCommunication		
interface component	*abbrev*	*provided by*
DsoCommunicationServer	**Ds**	DSO
DsoCommunicationClient	**Dc**	FP Node: only primary
Specifications:		
Ds		
1	***hand(DiscoverySpecification ds,Σ)***	
	To be determined ***received(DsoInfoId id, Σ)***	
Dc		
2	***shake(DiscoverySpecification ds,Σ)***	
	To be determined ***sendDsoInfo(DsoInfo s, Σ)***	

There is optional communication possible between the DSO and the node. This could involve the exchange of a structured distribution invoice.
Handshaking is realized by: the primary node using (1) and the DSO responding with (2).

4.1.4.5.4 T – ThirdPartyCommunication

Communication Interface		
ThirdpartyCommunication		
interface component	*abbrev*	*provided by*
ThirdpartyCommunicationServer	**Ts**	ThirdPartyInformationProvider
ThirdpartyCommunicationClient	**Tc**	FP Node: only primary
Specifications:		
Ts		
3	***receivedThirdPartyInfo(ThirdPartyInfoId id, Σ)***	
4	***hand(DiscoverySpecification ds,Σ)***	
Tc		
1	***sendSpecificThirdPartyInfo(DestinationIdentification d, SpecificInformationPackage si, Σ)***	

2	***sendGenericThirdPartyInfo(InformationFeed f, InformationFeedContent fcp, Σ)***
5	***shake(DiscoverySpecification ds,Σ)***

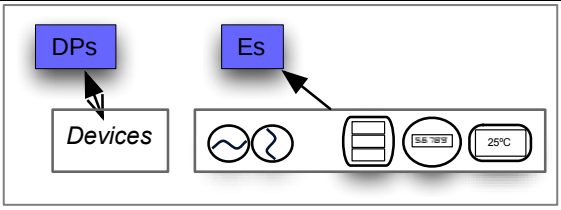
Third Party Information Providers can send their information to the node with (1) or (2), they can optionally receive a receipt back with (3). The Third Party Information Provider can be a specific server that sends information to a specific App. It identifies the App with a DestinationIdentification and supplies a specific information package that only the App can process. The DestinationIdentification must be a general App identification that is generic enough to span different versions of an App that all support the protocol. Alternatively, the FP Application Infrastructure platform can define categories of feeds of information, so that information providers can supply information of that category feed. Apps can subscribe to information feeds of that category.

As an example, the FP Application Infrastructure platform could define an information feed "weather forecasting info" and define a format for the information. ThirdPartyInformationProviders can then provide weather forecasting information using this format. Components in the FP Node could use this information by announcing that they subscribe to these feeds. These can be plugins or Apps.

The SMF of the node knows who subscribes to what feed, so it can direct the incoming information feeds to all components that have a subscription.

Handshaking is realized by: the primary node using (4) and the provider responding with (5).

4.1.4.6 *Communication Interfaces about Resources*



The communication interfaces are the connections to the physical infrastructure. They offer connections to the DeviceProtocols used to monitor/control the physical devices. They also connect to a smart meters to read out the general consumption and generation, or other smart meter information.

4.1.4.6.1 *DP - DeviceProtocolServices*

Communication Interface		
DeviceProtocolServices	*Communication interface to link the Node to a DeviceProtocol.*	
interface component	*abbrev*	*provided by*
DeviceProtocolServicesServer	**DPs**	A ProtocolConnector
	DPc	The specific device protocol
Specifications:		
DPs	Specific for a specific device protocol.	

4.1.4.6.2 *E – EnergyServices*

Communication Interface		
EnergyServices	*Communication interface to supply general consumption/generation to the Node and to provide sensor-information from additional meters.*	
interface component	*abbrev*	*provided by*
EnergyServicesServer	**Es**	SMF of Node
	Ec	metering device
Specifications:		
Es	Specific depending on the read-out capabilities of the metering instrument. To be decided.	

Note that also secondary nodes have this interface; they forward the info to the primary node so that its EnergyApp has a complete view on the consumption and generation of all of its locations.

An Energy Application of a Node receives energetic flexibility information from the monitored/controlled devices via its ResourceManagers.

However, it also needs to know the total consumption and generation of the account location(s) to take the devices into account that are not monitored / controlled as a Resource. Furthermore, the energetic flexibility exposed by a resource manager does not include the actual energy consumption of an appliance.

Therefore the SMF exposes an Es interface, to receive the total consumption and generation of the account location. It supplies this information to the Energy Application Layer of the Node. The information is also accessible for GUI Widget applications to present information to the User.

The EnergyServices interface supports in general the reading of (smart) meters. In this way, one not only can receive total consumption and generation info, but can also connect (smart) meters to collect ambient condition information. The icons in the figure below are used to depict the different meters connected to an FP Node. The table describes the icons and the meters they represent.

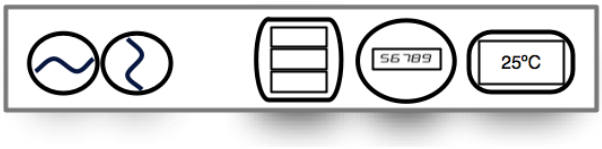
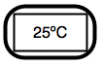


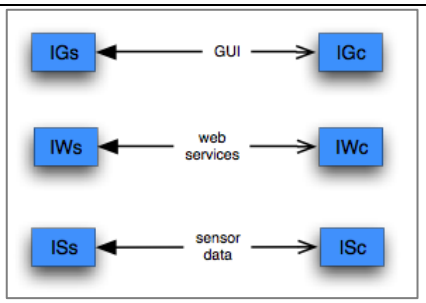
Figure 4-10 Smart meters for the EnergyServices interface

Type/Icon	Description
	Overall consumption and generation interval meter.
	Interval meter for specific appliance or appliance group.
	Temperature meter, typically for ambient conditions such as room or outside temperature.
	Other meters.

Note that the EnergyServices interface only deals with smart meters to read information. Other smart meters that provide control are considered smart devices and are accessed via the Ds interface and represented as Resources in the system. Other sensor information from User Devices (such as PC, smartphone or tablets) is also not subject for the EnergyServices interface but can be delivered to the system via the WebServices of the InteractionFramework.

In a technical realization, Apps, which provide information of an appliance or appliance group via smart-meter readouts, are METER apps of the RAL that provide the information to the FP Runtime services. In case the appliance itself and the protocol provides information readout capabilities, a ResourceDriver can also deliver information about its appliance.

4.1.4.7 *Communication Interfaces about Interaction*



There are communication interfaces for interaction with the Node using UserDevices. The IG interface is about the GUI provided by the Node; the IW interface is about the Web Services provided by the Node, the IS interface allows UserDevices to supply sensor information.

4.1.4.7.1 *IG – InteractionGui*

Communication Interface		
InteractionGui	*Communication interface to provide the GUI to control and interact with the Node.*	
interface component	*abbrev*	*provided by*
InteractionGuiServer	**IGs**	The INTERACTIONFRAMEWORK of the primary FP Node
InteractionGuiClient	**IGc**	UserDevice
Specifications:		
CGs	Not specified here, via Web server of INTERACTIONFRAMEWORK of Node	
CGc	Not specified here, via Web browser of user device	

4.1.4.7.2 *IW – InteractionWebservice*

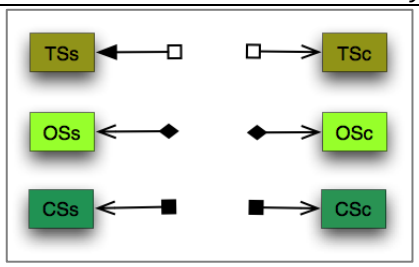
Communication Interface		
ControlWebService	*Communication interface providing general webservices to ask information from the Node.*	
interface component	*abbrev*	*provided by*
InteractionWebServiceServer	**IWs**	The INTERACTIONFRAMEWORK of the primary FP Node
InteractionWebServiceClient	**IWc**	Open to any
Specifications:		
CWs	Not specified here.	
CWc	Not specified here.	

4.1.4.7.3 *IS – InteractionSensor*

Communication Interface		
ControlSensor	*Communication interface to provide the Node with sensor data from UserDevices.*	
interface component	*abbrev*	*provided by*
InteractionSensorServer	**ISs**	The INTERACTIONFRAMEWORK of the primary FP Node
InteractionSensorClient	**ISc**	UserDevice
Specifications:		
CSs	Not specified here.	
CSc	Not specified here.	

UserDevices can supply sensor information (e.g. GPS, orientation information, etc.) to help the INTERACTIONFRAMEWORK enrich its GUI or to help decision logic in the Node. An example of the latter is supplying location information via smart phones of account members to help the decision logic.

4.1.4.8 *Communication Interfaces about System management*



These are communication interfaces about system management services, such as time synchronization, OpenServices and Configuration services.

4.1.4.8.1 *TS – TimeSynchronization*

Communication Interface		
TimeSynchronization	*Provides timeclock synchronization services to any in the FP Net*	
interface component	*abbrev*	*provided by*
TimeSynchronizationServer	**TSs**	FP ManagementCenter
TimeSynchronizationClient	**TSc**	FP Node: all (primary and secondary) Optional : FP AppStore, DSO, Supplier, EnergyMarketParticipant, ThirdPartyInformationProvider.
Specifications:		
TSs	Not specified here; typically via NTP; Serverside synchronizes with public service)	
TSc	Not specified here; typically via NTP; Client side NTP daemon uses server side)	

4.1.4.8.2 *OS – OpenServices*

Communication Interface		
OpenServices	*Interface to provide services management for installation and updates FP Node software, especially system, bundles, plugins and apps.*	
interface component	*abbrev*	*provided by*
OpenServicesServer	**OSs**	FP ManagementCenter
OpenServicesClient	**OSc**	FP Node: all (primary and secondary) FP AppStore
Specifications:		
OSs, OSc	Not specified here, relies on the OSGi services	

4.1.4.8.3 *CG – ConfigurationService*

Communication Interface		
ConfigurationService	*Configuration services to perform remote node management, especially service activation and reconfiguration, remote subscriber support, firmware and configuration management, diagnostics and monitoring.*	
interface component	*abbrev*	*provided by*
ConfigurationServiceServer	**CSs**	FP ManagementCenter
ConfigurationServiceClient	**CSc**	FP Node: all (primary and secondary)
Specifications:		
CSs	Not specified here. The CSs is provided by the FP ManagementCenter, typically as part of a TR-069 ACM (Auto Configuration Server).	
CSc	Not specified here. The CSc is provided by the FP Node device, typically regarded as TR-069 CPE (Customer Premise Equipment).	

4.2 Main Components Decomposition

4.2.1 Introduction

We provide a decomposition and more detailed functional description of the three main system entities in an FP Net:

- FP Node
- FP ManagementCenter
- FP AppStore

The external communications between these entities and other entities in an FP Net was already described in the previous section.

4.2.2 FP Node

4.2.2.1 Frameworks and Layers

The FP Node Software has 2 frameworks and 2 layers, as shown in the following figure. Frameworks and layers are both components of the FP Node software. We use the term layers

- To identify the components that are involved with the energy management decision logic, corresponding with the main FP Application Infrastructure concepts of Energy Application and Resources.
- To highlight the fact that layers are decoupled and independent, in the sense that they do not communicate directly with each other and that a layer is not aware of the specific composition of the other layers. The SMF manages the layers, their composition, their communication with other layers and their communication with external entities.

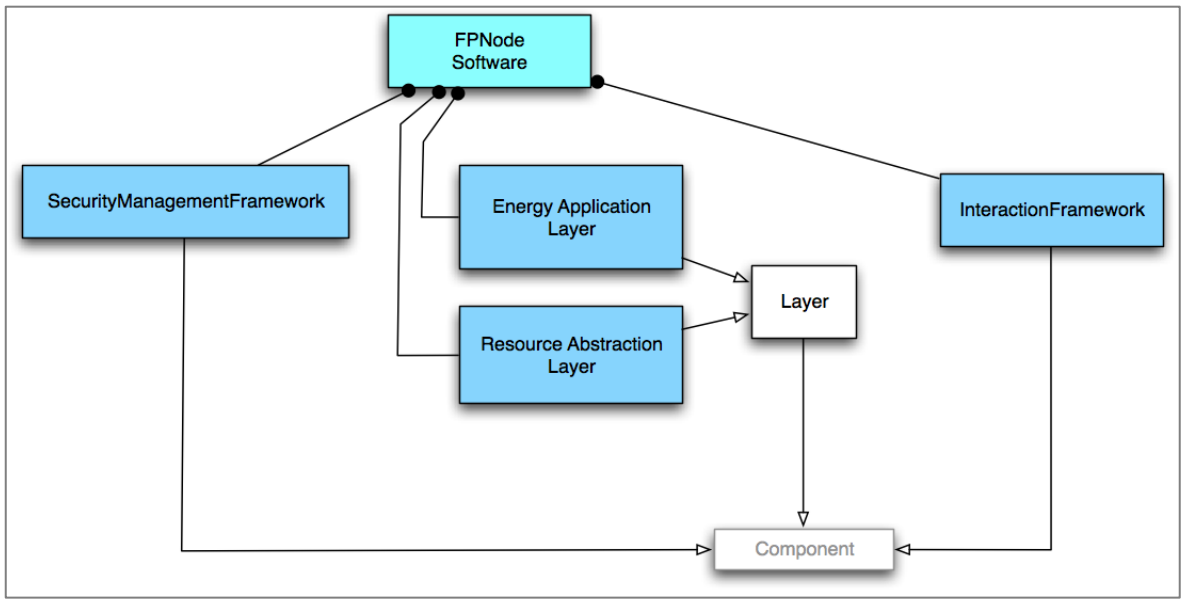


Figure 4-11 FP Node frameworks and layers

4.2.2.2 The SMF Model

The FP Node software has many data objects used by its software components to represent and exchange information. The Security Management Framework manages the FP Node, it therefore uses its so-called **SMF Model**, which is the collection of all important data objects needed to

represent the current state, to realize state transitions, to provide general storage and the artifacts to manage. In this section we introduce the different aspects of the SMF Model.

4.2.2.2.1 *WorldModel of representations*

An FP Node software system has many representations. With “representations” we refer to the data objects (instances) that are used by the software components to represent the current state of the Node.

We define the **WorldModel** as the collection of all representations that are automatically managed by the SMF, in the sense that the SMF ensures periodic syncing of these representations with the MC, so that the MC can use this information for monitoring and most importantly can use this information to restore state on re-installations and restore actions.

4.2.2.2.2 *EventLog of events*

Besides the representations that model the current state, there are artifacts, so-called “events” that are used to realize a state transition to the next state.

We define the **EventLog** as the collection of events that are either considered interesting for later analysis of state transitions, or are considered important to help with re-installation/restore actions. The EventLog is also subject of syncing operations with the MC.

4.2.2.2.3 *CoreStorage*

The management of internal state information of Plugins and Apps is not subject of management by the SMF. However, the SMF provides a so-called **CoreStorage** facility that Plugins and Apps can use to store information safely and persistent. This is a general storage facility that is also subject of syncing operations with the MC to provide persistency.

4.2.2.2.4 *Security Management model*

Besides the representations of the WorldModel, the events in the EventLog and the information of CoreStorage, the SMF also needs information to perform its management and security responsibilities. The collection of all of this data is called the Security Management Model or **SMModel**. This is about local management information only, not subject to syncing operations by the SMF to the MC.

4.2.2.2.5 *Examples of Information in the SMF Model*

The next table gives a (partial) overview of the different representations with some examples. It also marks which are part of which SMF model.

Representation	*Examples*	*Part of SMF Model?*
Abstract representations of External entities	Abstract representations of its external parties such as MC, AS, EnergyMarketParticipants, etc.	SMModel
Communication artifacts	Certificates, security policies, encryption keys, …	SMModel
Management artifacts	Authorizations, policies, time-clock, knowledge of what is already synced with the MC, etc.	SMModel
Decision object representations	ControlSpaces, Allocations, Energy Proposals, Energy Assignments, etc.	WorldModel
Information feeds	Information items received from ThirdPartyInformationProviders	WorldModel
User target settings	Static energy targets and dynamic energy targets	WorldModel
Account information	Users, user devices, roles and rights	WorldModel

	User sensor data (orientation/location info from user devices) Smart meter sensor data (total consumption/generation, ambient condition information such as home or outside temperature, etc.)	
Current configuration	Information (version info) of the currently installed PMSuite software; application Plugins, Apps and AppGroups; application components (ProtocolConnectors); software bundles (versions of its components and/or layers should one decide to make some of these updatable bundles),	WorldModel
State of Layers and their communication	Messages being exchanged, temporarily present in queues during communication flows, …	EventLog
Internal state of Plugins and Apps	Internal representations of the decision logic of an Energy Application, internal representations made by an App for its reasoning, internal representations of a ResourceManager, state information during interactions of a GUI, ...	no
State information that a Plugin or App wants to have persistent	If a plugin on app wants to have information made persistent, in the sense that it will safely survive node failures and will be used in reinstallation/restore operations, it can store this information in a CoreStorage facility of the SMF.	CoreStorage

4.2.2.2.6 SMF Model diagram

The WorldModel, the EventLog and CoreStorage are all managed by the SMF and subject to (incremental) sync operations with the MC.

- The World Model has a Decision Object Model, which not only has the DecisionObjects (such as ControlSpaces, etc.) but also DecisionObjectRelations. These are the relations between the Decision Objects, for example the relation between an EnergyProposal and the ControlSpaces it was based on. This separation between decision objects and their relations is important, since an EnergyProposal itself should not contain information about underlying ControlSpaces. A ResourceManager knows about its ControlSpace since it created it to expose its energetic flexibility, but it does not know about EnergyProposal made to cover its ControlSpace. An Energy Application on the other hand knows the relation between EnergyProposal and ControlSpaces. We will explain this further when we introduce WorldViews.

- The Account Model is part of the WorldModel. The information of the Account, its Users, its User Devices and the assigned Roles all can be more accurate than the account information of the MC, therefore the Account Model is part of the WorldModel. The definition of which Rights belong to a Role cannot be more accurate since managed by the MC; therefore that information is not part of the Account Model but of the SMModel.

- Note that there are some parts of the WorldModel that do not require syncing to the MC. This is the case for the Sensor Model of the Account Model. It contains user sensor data of UserDevices (location/orientation information etc.) and smart meter sensor information (total consumption/generation, ambient information such as home or outside temperature, etc.) It is also the case for third party information as part of the decision object model. The decision objects representing the information availability should be synced but syncing their contents is not required. The level of syncing is determined by a policy.

The next figure shows the functional class diagram of the SMF Model. The models in green are subject to syncing operations; with only partial syncing requirements for their parts with dotted borderlines.

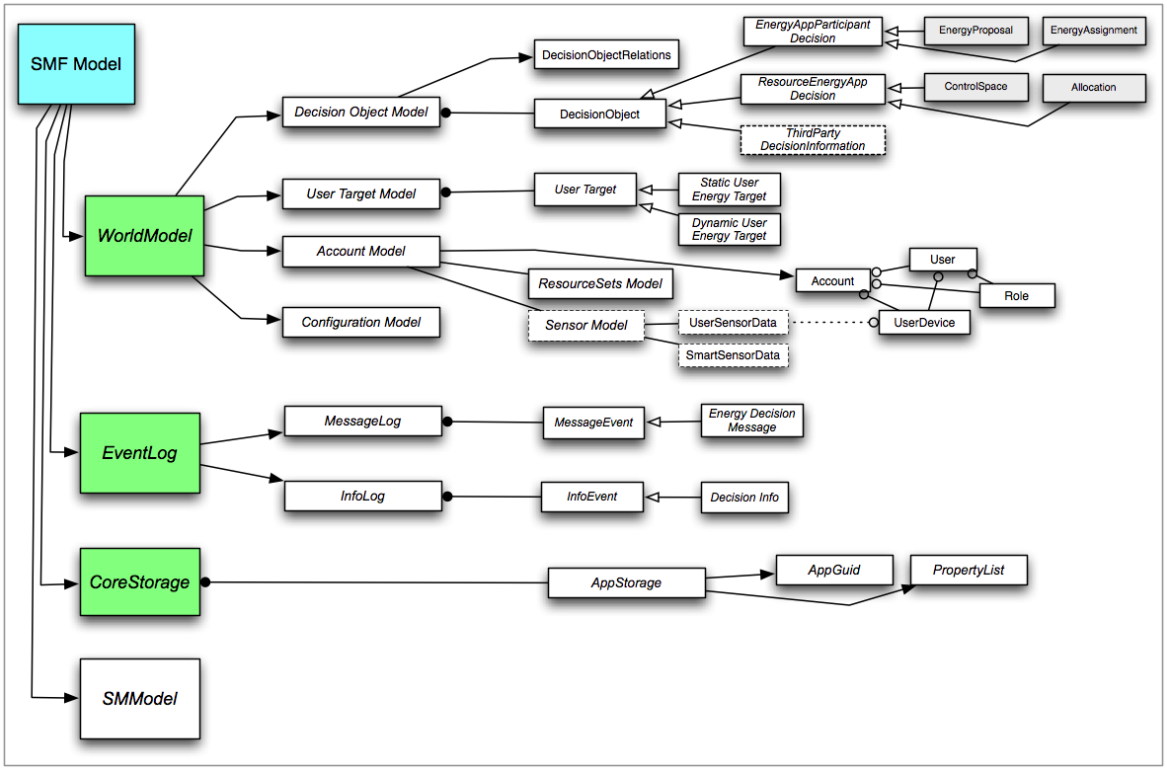


Figure 4-12 The SMF Model

The EventLog contains a MessageLog that has, as one particular kind of MessageEvents, so-called Energy Decision Messages. This will be explained in the next sections. Note for now that the MessageLog and the InfoLog is an ordered set of items, with surrounding information to manage its syncing with the MC.
The CoreStorage contains for each App (or Plugin) a PropertyList. A PropertyList is a feature-value structure that provides a simple structure to store information.[29]

The SMModel, not subject of sync operations with the MC, has the following structure.

[29] Note that the terms CoreStorage or PropertyList do not refer to the technical terms CoreData and PropertyLists as available in the iOS development environment, although these could inspire a good technical design.

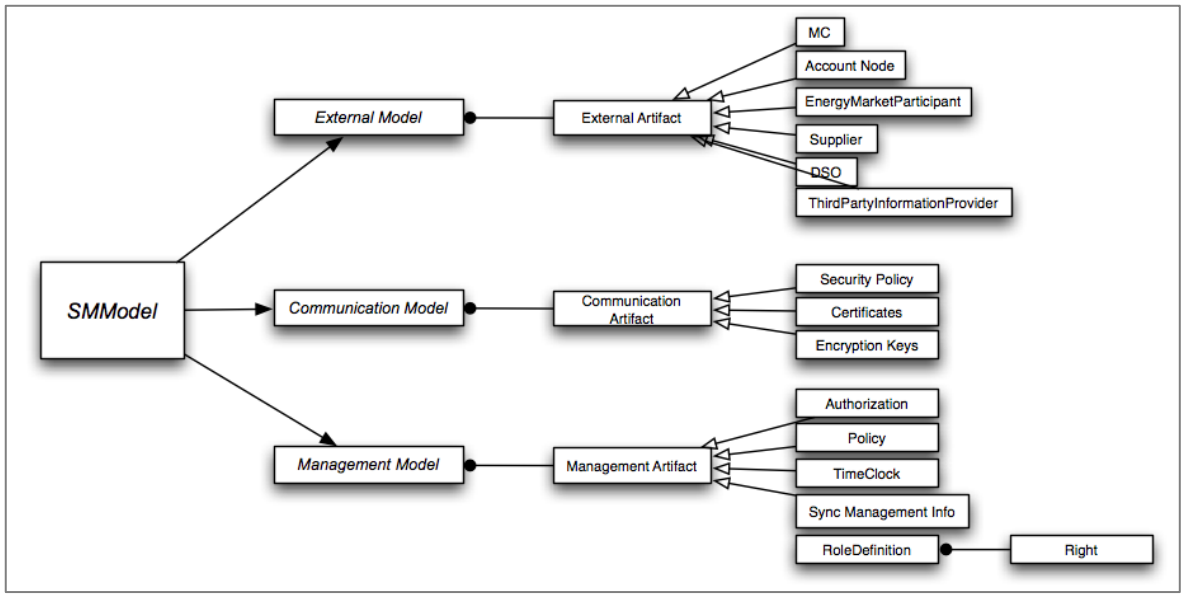


Figure 4-13 The SMModel of the SMF Model

The SMModel contains

- An external model with external artifacts that are data representations of all external parties involved in communication, useful in a technical realization to contain addressing information for the external communication interfaces of the Node.
- A communication model with artifacts related to communication actions. These include security policies, certificates, encryption keys etc.
- A Management Model with artifacts needed by the SMF to perform its management. This includes authorizations, policies, the time-clock (synchronized with the MC), all management information the SMF needs to keep to perform its sync operations (with MC as primary node, or with the primary node as secondary node). It also contains the definitions of Roles, stating which access rights a role covers.

When the SMF is shown in decomposition diagrams in the next sections, we will use the following figures to display the SMF Model:

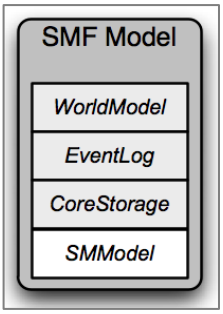


Figure 4-14 Icon for the SMF Model

Figure 4-15 Small Icon for the SMF Model

4.2.2.2.7 *SMF Model realizations*

Although the SMF Model is a functional concept, it can serve as a good perspective for an efficient technical realization.

One technical realization possibility is to let the SMF manage all instances of important objects in the Node as part of its SMF Model. When pieces of the Node software create instances, they could make use of factories of the SMF. When the SMF Model contains all instances, it is not only

easier to efficiently organize its syncing operations, but more importantly it can ensure a simple and efficient way of managing references.

Since the Node represents a dynamic system with objects representing an external reality, the technical realization should make sure that it provides an efficient manner to deal with references to objects that suddenly are no longer valid because the external reality changed.

As an example, disconnecting a physical device should (maybe after some time or bridging short temporarily disconnects) make the corresponding Resource and its ResourceManager invalid and further up make ControlSpaces and EnergyProposals depending on it invalid or indicate that they are no longer fully reflecting reality. Decision structures or intermediate messaging artifacts need to be able to refer to invalid objects. Also one needs a way to keep referring to objects that no longer live in the Node but are already synced with the MC and no longer needed by the Node since outside of the current management window.

One therefore could have all instances be part of the SMF Model, created by SMF factories and only accessible under strict SMF control. All other entities of the Node (layers, plugins, apps, etc.) could instead use reference objects that do not have pointer references but an abstract way of representing a reference. This way, the SMF could clean up instances while still the other Node entities could have references in their local representations. Technically this would mean that the SMF has instances with full access getters/setters and so on, while others only have (via interfaces) a reference objects with restricted interfaces that underlying work via a SMF controlled way on the real instances. Besides these references, there can be of course many real instances present in Plugins, Apps, etc., but these are not subject to SMF management.

4.2.2.3 *Communication Schemes*

4.2.2.3.1 *Levels of Communication*

An FP Node communicates with external entities and has internal communication flows between its components and subcomponents.

We distinguish different levels of communication:

- Energy
- Interaction
- Management
- System

The energy management related communication is the most meaningful with respect to the energy functions of the Node. It is therefore important to properly represent, support, monitor and log this communication.

The other levels of communication are important for the correct interaction, management and system operations of the node, but as a design principle we clearly distinguish the energy management level from the others and consider it as being a different “plane” of communication, different from the others.

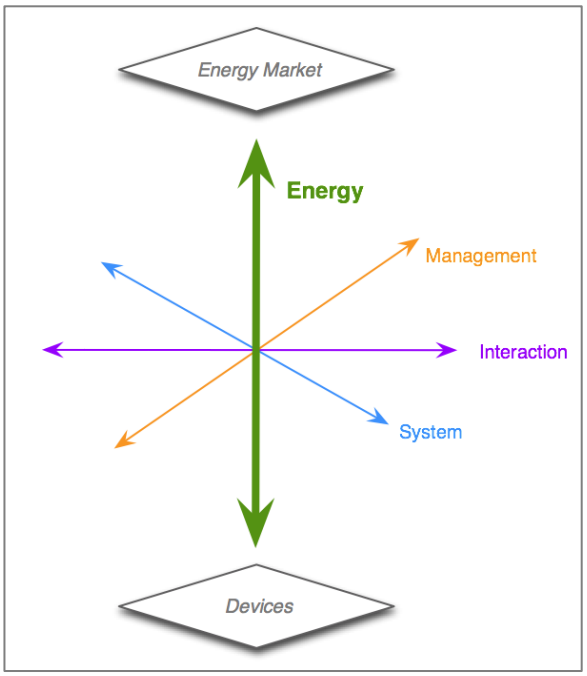


Figure 4-16 Energy communication versus others

The FP Nodes support these different levels of communication with different so-called **Communication Schemes**.

4.2.2.3.2 *Communication Schemes to realize communication levels*

A **Communication Scheme** is a proscribed method of managing the interactions and information flows between external communication interfaces and internal components, to support a level of communication. A Scheme consists of a set of representations actors and action sequences.

Communication Scheme	*Abbreviation*	*Communication level supported*
Energy Communication Scheme	ECS	Energy
Interaction Communication Scheme	ICS	Interaction
Management Communication Scheme	MCS	Management
System Communication Scheme	SCS	System

The following tables show the different schemes, which of the external communication interfaces are involved and what FP Node components are involved. In the next sections we will describe the schemes in more detail.

Communication Schemes for a primary FP Node:			
Scheme	*External communication interfaces*	*Components involved*	*Managed By*
ECS	Pc, DPs, Es, Tc, F1 (*)	SMF, EnergyApplicationLayer, ResourceAbstractionLayer	SMF
ICS	ASc, IGs, IWs, ISs	Interaction Framework	Interaction Framework[30]
MCS	CGc,CTc Sc,Dc,Tc MIc,MAc,MPc,MRc,MMc,MUc MI1,MA1,MP1,MR1,MM1 (*)	SMF	SMF
SCS	TSc, OSc, CSc	SMF, system[31]	SMF, system

In case there are secondary nodes involved, also the interfaces marked with (*) in the previous table are involved and a secondary node realizes the schemes according to the next table.

Communication Schemes for a secondary FP Node:			
Scheme	*External communication interfaces*	*Components involved*[32]	*Managed By*
ECS	DPs, Es, F2	SMF, ResourceAbstractionLayer	SMF
MCS	CGc,CTc MI2,MA2,MP2,MR2,MM2	SMF	SMF
SCS	TSc, OSc, CSc	SMF, system	SMF, system

A secondary node does not have a ICS, since it has no interaction provisions. However interactions affecting the secondary node are forwarded via F1/F2. As an example, a user using the GUI to change dynamic user targets for a remote device (device managed by a secondary node), interacts with its GUI via the InteractionFramework of the primary node, but as a result the

[30] The ICS is managed by the InteractionFramework itself, not by the SMF. The ICS has a token-based authorization to do this obtained from the SMF.
[31] Underlying software & firmware system (OSGi framework, JVM, OS, TR-069 services).
[32] Besides the components of the secondary node that are involved, of course also the components of the primary node are involved, since the secondary SMF interacts with the primary SMF to realize the communication scheme.

dynamic user target settings will end up in the resource manager of the secondary node, since that remove device is managed by that resource manager.

Most of the Communication Schemes are managed by the SMF of the FP Node. In case there are secondary nodes, the SMF of the primary Node manages the overall scheme and interplays with the secondary SMF's that manage it locally on their secondary Node.

In the next sections we will describe the Communication Schemes on a functional level, where it typically consists of representation definitions, functional queues of messages, descriptions of sequences of actions. A technical realization can typically consist of data models definitions, persistence methods, technical queues, consumers/producers, concurrency provisions and so on. As long as the technical realization covers the functional definition, the specific technical design choices remain open. Note that the functional descriptions include some aspects of asynchronous messaging, but keep the requirements minimal for proper functional design to allow for a lightweight technical realization. For example, although there is a functional need for asynchronous messages and some queues to decouple components and information flow, we did not assume a full-fledged asynchronous message bus structure since not all aspects are functionally required.

4.2.2.3.3 *External Communication Interfaces of an FP Node*

An FP Node has a series of external communication interfaces, described in the previous section, shown in the next figure.

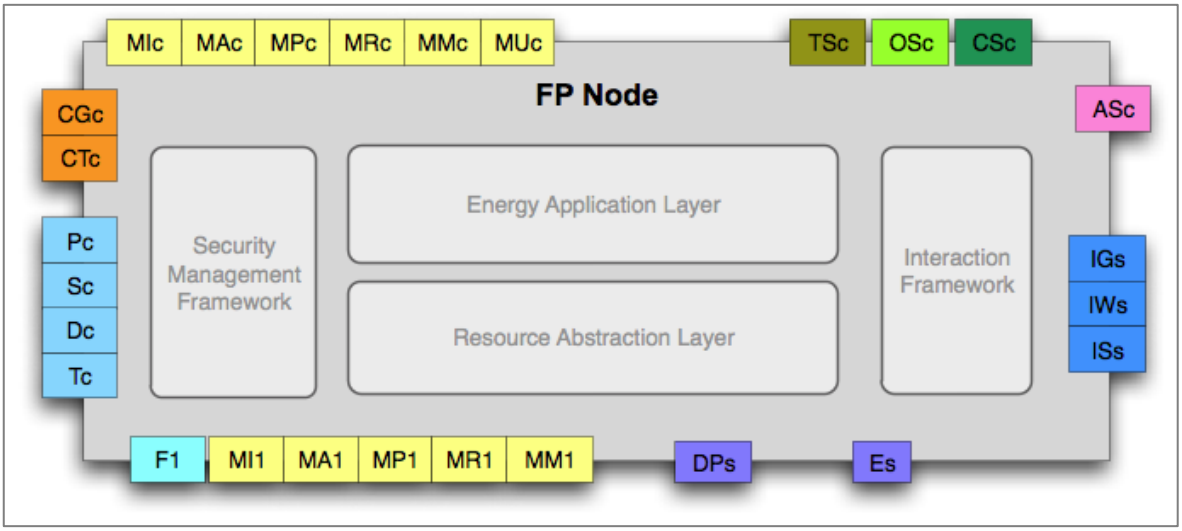


Figure 4-17 FP Node with external communication interfaces

We will focus in the decomposition mainly on a primary FP Node. Recall that a secondary Node is highly similar, as shown in the next figure.

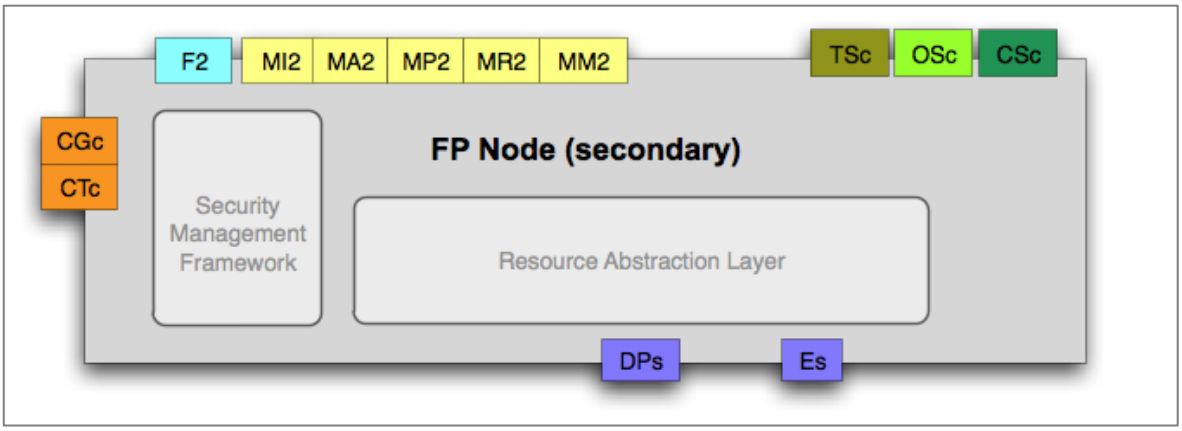


Figure 4-18 Secondary FP Node with external communication interfaces

A secondary Node only has a SMF and a ResourceAbstractionLayer, but no EnergyApplicationLayer or InteractionFramework. The SMF of a primary Node makes the presence of secondary Nodes to monitor/control remote devices transparent for its layers. A primary Node communicates with the MC, a secondary Node uses highly similar communication but talks with its primary node instead, which almost completely takes over the responsibility as management center for the secondary nodes. A secondary Node has no interfaces for external parties, it has no interaction interfaces or store interfaces.

4.2.2.4 *General Communication provisions*

Before continuing the FP Node decomposition and the examination of the communication schemes in more detail, we already can observe a number of subcomponents in the following diagram.

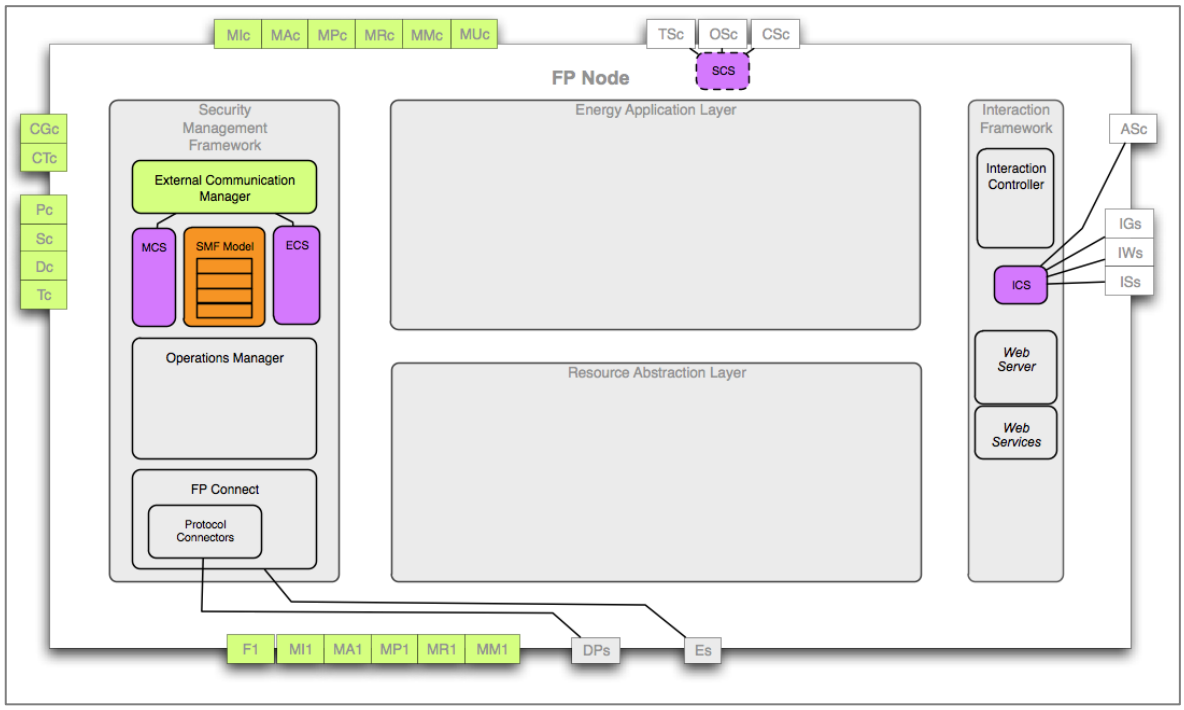


Figure 4-19 General provisions as subcomponents in the Node

The SMF, responsible for security and management, uses several external communication interfaces. These interfaces (shown in green) are connected to a so-called **External Communication Manager**. This subcomponent is responsible for queuing incoming messages from the external communication interfaces, authentication by certificate verification, decryption, destination resolution and routing[33]; and for outgoing traffic for queuing outgoing messages, provision of the certificate, encryption, destination resolution and using the external communication interfaces to send the info. Its counterpart is the **FP Connect**, interacting with device protocols and reading total energy consumption/generation values or other (smart) meter information. The **ECS** and **MCS** communication schemes interact with the External Communication Manager. The SMF also has the **SMF Model** introduced earlier. The SMF has an **Operations Manager**, responsible for local management including syncing the information with the MC (or in the case of a secondary node syncing with the primary node), an **Authorization Manager** and a **Policy Manager**.

The **SCS** communication scheme is not really part of the SMF since situated more on a system level.

The ICS scheme is part of the InteractionFramework, which further consists of an **Interaction Controller**, a **Web Server** and **Web Services**.

[33] For incoming traffic, routing is about examining DestinationIdentifications, InformationFeeds, AppGuids and so on.

4.2.2.5 *Energy Communication*

The “energy” communication stream is about the message flow related to energy management, involving EnergyMarketParticipants, Energy Applications, Resources and ThirdPartyInformationProviders.

4.2.2.5.1 *High-level energy information flow*

The high-level functional view of the energy information flow is shown in the following figure.

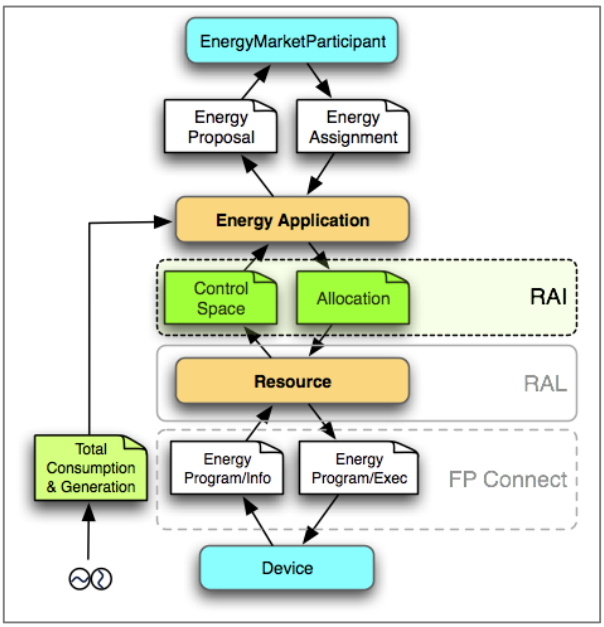


Figure 4-20 High-level functional view on energy information flow

Devices provide energy program/info and are represented in the RAL of the FP Node as Resources.

The total energy consumption/generation is also provided to an Energy Application.

A Resource communicates its energy information as a ControlSpace to the Energy Application.

The Energy Application makes an EnergyProposal for its EnergyMarketParticipant. These participants communicate EnergyAssignments back to the Energy Application which makes Allocations for the Resources to cover their ControlSpaces. Resources provide program execution info to the physical devices.

Note that the “Resource Abstraction Layer” principle from the first chapter results in an abstraction of device specifics into Resources and a boundary between Energy Application and Resource of generic objects: ControlSpaces (exposing energetic flexibility) and Allocations. These are specified by the **FP RAI** specification of the FP Application Infrastructure.

The other boundaries are not generic, but specific. They are realized by container objects such as EnergyProposals and EnergyAssignments. These have a non-generic payload specific for the EnergyMarketParticipant (and understood by the Energy Application). Note that total energy consumption/generation info is represented in a generic way; for these the specifics of smart meters and the specific format of their readouts is translated into a generic representation by the external communication interface (Es). Note that the boundary between a Resource and its Device is not realized necessarily by container objects, rather by the ResourceManager using the interface of a ResourceDriver that knows the specifics to interact with the Device via a DeviceProtocol. The connection of devices with protocols and the FP Node is specified in the **FP Connect** specification of the FP Application Infrastructure.

This communication flow is realized by an interaction between de Node layers, the SMF and the communication interfaces.

4.2.2.5.2 *Energy Communication Scheme (ECS)*

The next figure shows the Energy Communication Scheme (ECS) of the FP Node.

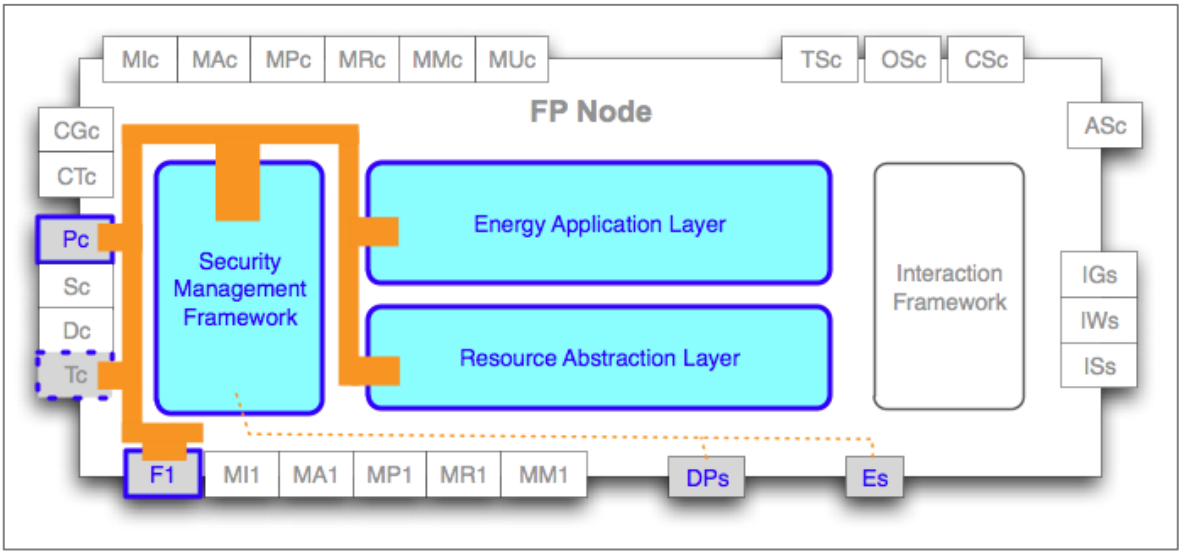


Figure 4-21 Energy Communication Scheme (ECS) of a primary Node

The SMF plays a central role. With DPs, it provides the connection to the DeviceProtocols talking to the physical devices. With Es it is able to read the overall consumption and generation, it makes this available to the Energy Application Layer so that an Energy Application has a complete view. The communication with EnergyMarketParticipants is realized via the Pc interface. Energy-related information from ThirdPartyInformationProviders enters via Tc. Other third-party information can enter via Tc as well. The communication interface F1 is used in the case there are secondary Nodes (see further).

We will first describe how the high-level functional view on the energy information flow is translated using the components involved in the ECS.

Later we come back to the properties and organization of the ECS.

4.2.2.5.3 *Energy communication flow in detail*

The next figure shows how the high-level energy communication flow is executed by the FP Node infrastructure, for the simple case of Resources for Devices local to the node.

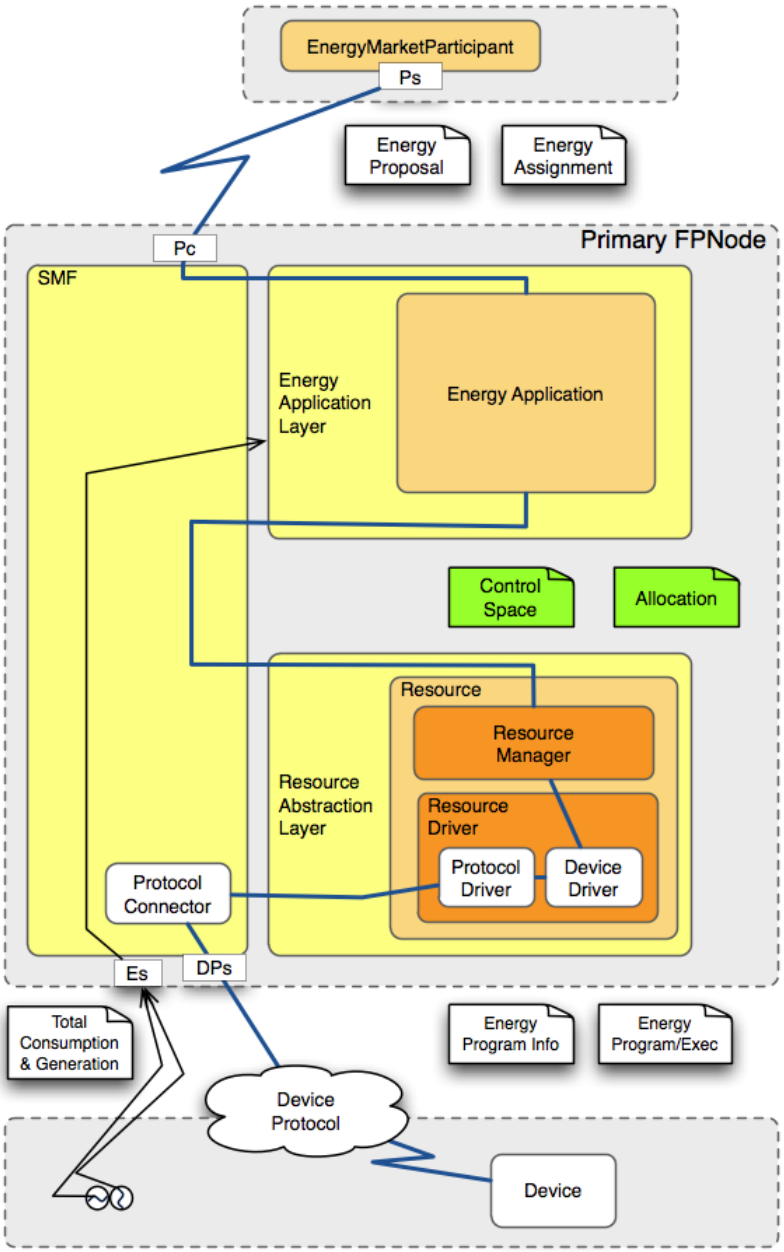


Figure 4-22 Energy communication involving SMF and layers

An Energy Application lives in the Energy Application Layer, a Resource lives in the Resource Abstraction Layer.

Communication between Resources and Energy Application(s) all happens via their respective layers that interact with the SMF.

The next figure illustrates the case where there are secondary nodes. The interfaces F1 and F2 are involved for communication between the primary and secondary node.

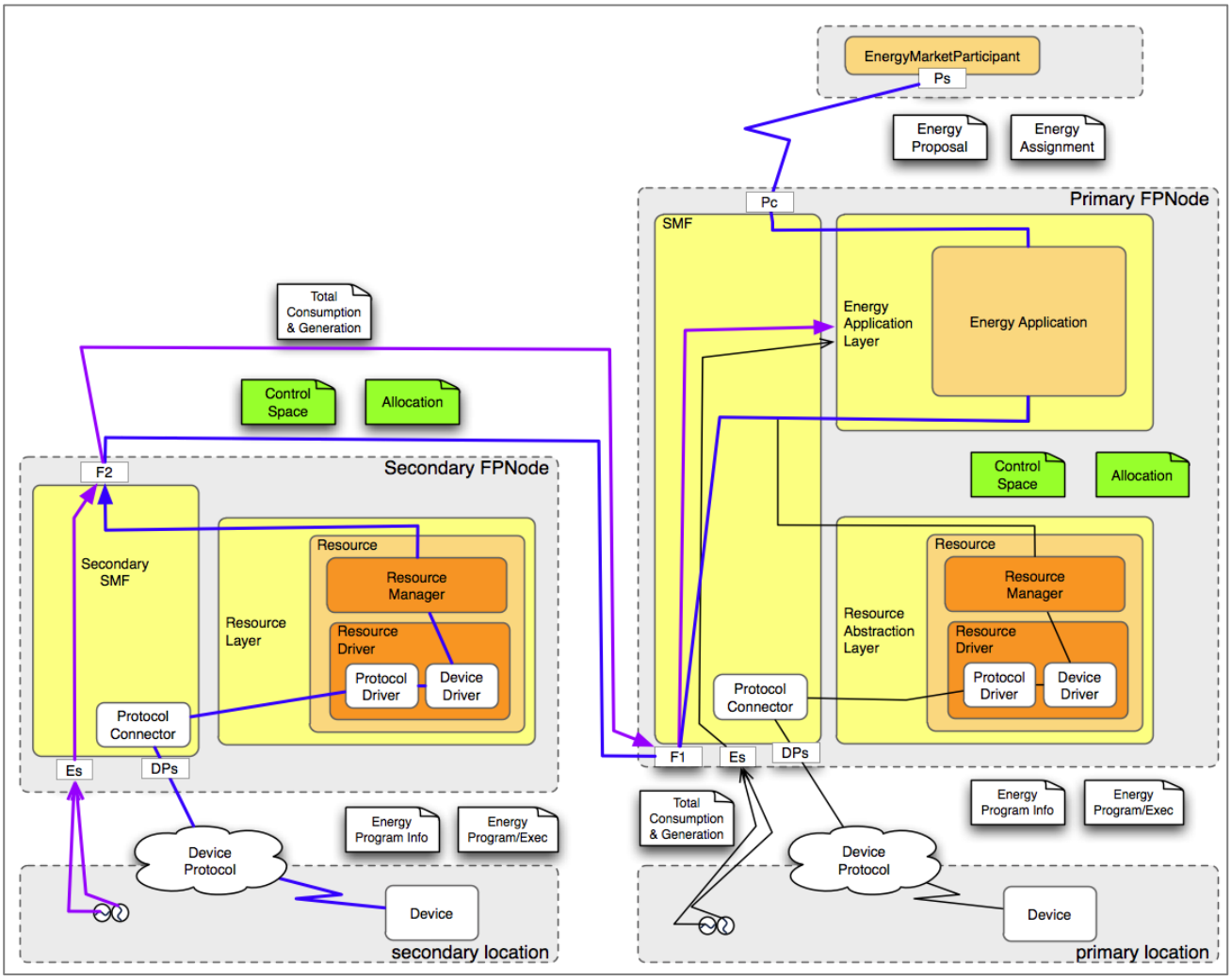


Figure 4-23 Energy communication involving a secondary node

4.2.2.5.4 *Properties and organization of the ECS*

The ECS ensures a decoupling of the energy management layers, so they are independent, shields them from external communication complexities, providing transparency for the location of devices and puts the SMF in charge, so that important messages and representations are automatically preserved and updated.

4.2.2.5.4.1 *Overview of the ECS*

The scheme is depicted in the next figure, where the ECS provisions on the diagram are shown, together with two other SMF subcomponents, namely the External Communication Manager and the SMF Model, since both are involved in the ECS operation.[34]

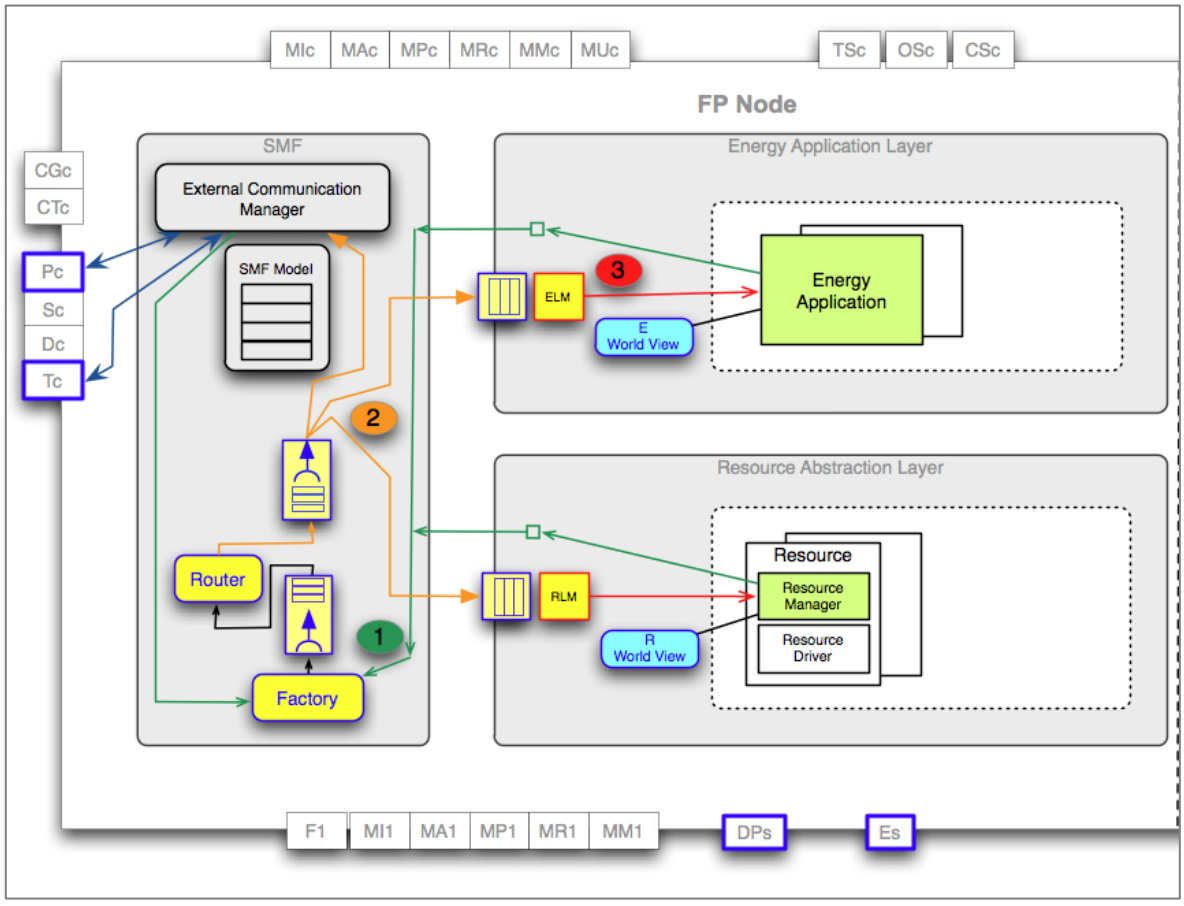


Figure 4-24 Energy communication scheme details

The initiators of energy communication are the EnergyApplication, Resource Managers and EnergyMarketParticipants. ThirdPartyInformationProvider providing energy-related information are also considered initiators.

We make abstraction of the communication with physical devices via DeviceProtocols. It was already described that the ProtocolDriver of a ResourceDriver of a Resource connects to the DeviceProtocol using a ProtocolConnector of the SMF. The information flow by protocol communication to and from devices is not subject to preservation and syncing with the MC.

[34] The SMF has more subcomponents than shown here, but only two are shown together with details of the ECS.

We also make abstraction of the information flow to measure the total consumption and generation via the Es interface. This information, together with the forwarded information from secondary nodes, is combined in the SMF, to supply the total consumption and generation information to the EnergyApplicationLayer.

4.2.2.5.4.2 A 3-step process

The energy communication flow in the ECS is a 3-step process. In the previous figure the steps are labeled and the arrows are shown in different colors.

Step (label)	*What*	*Description*	*Arrows (color)*
1	Submitting decisions	An initiator (EnergyApplication, ResourceManager, EnergyMarketParticipant, ThirdPartyInformationProvider) submits information	Submit call (Green)
2	Routing decisions	The SMF routes the information to the appropriate destination	Delivery to the destination mailbox (Orange)
3	Receiving decisions and act	A decision maker (EnergyApplication, ResourceManager) is triggered to use the information for its decision making.	Delivering the information and notification of arrival to decide further (Red)

We discuss the steps in more detail in the following paragraphs, where we refer to the previous diagram and explain the scheme provisions.

4.2.2.5.4.3 Step 1, submitting decisions

In step [1] decision information is submitted.

a) A ResourceManager decides to make a ControlSpace
b) The EnergyApplication decides to make a EnergyProposal, or decides to make an Allocation
c) An EnergyMarketParticipant makes an EnergyAssignment. Note that we do not have the real decision point here, but we consider the arrival of an EnergyAssignment at the Pc interface as the decision point.
d) A ThirdPartyInformationProvider provides energy-related information. The arrival of the information package at the Tc interface counts as decision point for providing the information. Only energy-related third-party information from the Tc interface is processed by the energy communication infrastructure. As already indicated, the SMF is capable of determining this, based on the DestinationIdentification or the InformationFeed.[35]

- For the first 3 cases, (a, b, c), the initiator makes use of a function of its layer to announce its decision object.

For example, a ResourceManager making a ControlSpace c,

[35] When a DestinationIdentification is used, the SMF can determine if it concerns destinations related with energy management, such as ResourceManager, ResourceManagerHelper, EnergyApplicationHelper, etc. As an example, weather-information used by a ResourceManagerHelper is energy related and enters the ECS. Also when information comes in as an InformationFeed, the SMF can determine if its energy related, since if energy-related Plugins/Apps (again, ResourceManager, ResourceManagerHelper, EnergyApplicationHelper, etc.) subscribe to such a feed, the information feed package will enter the ECS.

```
ControlSpace c = makeControlSpace(…)
```

can submit it using something like the following, where e is an expiration time (see further).

```
getLayer().submitControlSpace(c,e)
```

In a similar way when we have an EnergyApplication making an EnergyProposal taking into account ControlSpaces cs1,cs4,cs7 from all available ControlSpaces, with

```
EnergyProposal p = makeProposal(…,cs1,cs4,cs7)
```

Then the EnergyApplication can submit the proposal p[36] with

```
getLayer().submitProposal(p)
```

Please note the following on the above:

- When an EnergyApplication as in this example makes a decision, the SMF knows what all the available ControlSpaces are to base a decision on, since its ECS controls the submission system of ControlSpaces and also (see step 3) controls the trigger for decision making.
- Constructing a Proposal as shown with makeProposal(..) should under the hood make use of the SMF for the construction, so that the SMF can construct an EnergyProposal instance as part of its WorldModel (part of its SMF Model). The SMF can also keep track of the relation between the new EnergyProposal and the ControlSpaces it was based on and all the ControlSpaces that were available when making the decision. This info should not be part of the EnergyProposal itself, but the WorldModel can keep these relations.
- Note that for generic objects such as ControlSpaces and Allocations an SMF factory could construct the entire object. Specific objects such as an EnergyProposal are represented by container objects, for which an SMF Factory can construct the container object but the EnergyApplication has to construct the specific payload.
- For a ResourceManager using an Allocation, an Allocation is not the real DecisionObject but an interface that provides a reference to the DecisionObject and its relations with other DecisionObjects that are important to know for the ResourceManager. The ResourceManager is not allowed to know the relation of the Allocation with the EnergyProposal it was based on. We will elaborate further on this when introducing WorldViews.

The layer will use the SMF and its FACTORY (see diagram) to make a representation of this event. We call this an ENERGY DECISION MESSAGE (EDM) or briefly a “message”. This message is logged in the MESSAGELOG (part of SMF Model) and put on an in-queue of energy messages.

- For the last 2 cases (d,e), the external communication information received by Pc or Tc, is handled by the EXTERNAL COMMUNICATION MANAGER component of the SMF. For energy related information, being everything from Pc and some from Tc, it also uses the “Factory” to represent a “message”. Again, the Factory logs the message in the “Message Log” and adds it to the in-queue of energy messages.

There is an important difference between “making a decision” and “submitting” a decision.

The first, for example by

```
EnergyProposal p = makeProposal(…,cs1,cs4,cs7)
```

makes (and creates) a decision object.

The second, for example by

```
getLayer().submitProposal(p)
```

submits a decision.

It is important to separate these to allow for a decision maker to make decisions, review and maybe reconsider them and only submit them if the decisions are final.

This aspect is important for example for an EnergyApplication, since it is supposed to submit a consistent set of EnergyProposals. By this separation it can for example implement its decision

[36] Note that if desired the SMF is able to determine the expiration time from this, since it knows about the relation between f and the underlying ControlSpaces, it also knows about the expiration times for each of the ControlSpaces.

logic by examining the ControlSpaces available, check which of those are not covered by Assignments (and Allocations) already, then make EnergyProposal decisions for the ones it wants to cover, then review these new decisions by comparing them for overlap and comparing them with past proposals and received assignments, to only finally take its decision as a set of EnergyProposals by submitting them.

We hereby introduced the following subcomponents:

Symbol	Subcomponent	Responsibility
	Function in Layer	Function in Layer to submit a decision object
	FACTORY	Factory in the SMF that creates energy "messages", adds them to the Message Log and adds them to the In-queue of energy messages.
	IN-QUEUE ENERGY MESSAGES	A queue of energy messages where incoming messages are queued. Note that this queue is an essential component that serves not only as a landing strip for internal ECS messages and the mix with messages from external communication interfaces, but more importantly also as the landing strip for ECS messages from secondary nodes. The "landing strip" combines asynchronous behavior of different nodes.[37]

We also introduced the Energy Decision Message:

Data Object	Representation for
ENERGY DECISION MESSAGE (EDM)	An EDM is a representation of an energy-related decision or piece of information that is important for energy management. It is the registration of an event of submitting, routing or receiving decision information. It contains source and destination information, timestamp information, a reference to the decision object itself (e.g. ControlSpace) and some more information (see further).

This first step is designed to collect decisions in a general way, whether they originate from internal decision makers or external parties. It also makes sure that these decision objects can be logged in a general way and that they can be used for restoring a Node in a given state. Recall also from the energy communication flow details discussed earlier, that the in-queue will also receive EDM's from secondary nodes, so that the ControlSpaces and Allocations from remote Resources are processed correctly. We will elaborate later on the specific structure of an EMD, but first introduce the 2nd step.

4.2.2.5.4.4 Step 2, Routing decisions

In the 2nd step, a ROUTER component of the SMF, responsible for routing EDM's, picks EDM's from the IN-QUEUE, examines their source and destination information and consults its configuration information to determine the specific destination. The configuration information consulted is part of the Configuration Model, which is part of the SMF Model. It creates a new EDM with this specific destination information, adds it to the MESSAGE LOG and adds it to an OUT-QUEUE of energy messages.

The EDM's that are put on the out-queue are directly delivered for further processing.

- EDM's for an EnergyApplication are delivered to the MAILBOX of the EnergyApplicationLayer.

[37] Internally for one Node, one could think about a design that works fully synchronous. But primary and secondary nodes do not work in one synchronous cycle, therefore there is an asynchronous behavior and a need for a "landing strip" to manage everything. Moreover there is asynchronous behavior from external parties as well, such as EnergyMarketParticipants and ThirdPartyInformationProviders.

- EDM's for a ResourceManager are delivered to the MAILBOX of the ResourceAbstractionLayer. This could also imply delivery to EDM to forward it via the F1-F2 external communication interfaces to a secondary node.
- EDM's for an EnergyMarketParticipant are delivered to the ECM component of the SMF that will handle the external communication via the Pc interface.

The so-called MAILBOX component of a layer serves the functional purpose of assembling all decision information, which the next decision logic cycle can take into account. This design wants to separate the arrival of new decision information from the act of triggering a new decision logic cycle to take this into account. In the 3rd step, a specific trigger will launch the decision cycle.

Please note that routing an EDM can trigger the creation of multiple EDM's. For example, when routing an EMD for an information package from a ThirdPartyInformationProvider that has an InformationFeed specified, the information should be delivered to all apps that subscribed to this feed. In that case, the single EDM from step 1 containing the information package as submitted by the ThirdPartyInformationProvider (entering the ECS via the External Communication Manager reading an external communication message from the Tc external communication interface).

We hereby introduced the following subcomponents:

Symbol	Subcomponent	Responsibility
	OUT-QUEUE ENERGY MESSAGES	A queue of energy messages where outgoing messages are queued.
	ROUTER	A component of the SMF that is responsible for picking EDM's from the in-queue, determining their routing information and creating a new routed EDM and adding it to the out-queue.
	MAILBOX	Each of the layers has a Mailbox to collect decision information on which the next decision logic cycle can act.

4.2.2.5.4.5 Step 3, Receiving decisions and act

In the 3rd step, the decision logic present in components of each layer is triggered to execute a new decision cycle, taking into account all available decision information. For this purpose, each layer has a **Decision Logic Manager** and a **WorldView**.

- A Decision Logic Manager triggers decision logic cycles of all decision components in a layer.
- The WorldView of a layer provides the decision logic components of that layer with the correct view on all information to make their decision.

The next figure shows the types of Decision Logic Managers and the types of WorldViews.

Figure 4-25 Decision Logic Managers and WorldViews

In the next table, we identify the ECS artifacts for each layer and describe their responsibilities. We will elaborate further on WorldViews in the next paragraph.

Layer	*ECS artifacts for Layer*	*Responsibility*
Resource Abstraction Layer	Resource Logic Manager (RLM)	Manage the decision logic cycles of its decision logic components, being the ResourceManagers of its Resources
	Resource World View	Provide a correct view for a ResourceManager[38] on all information it needs to make a decision. It needs to know for example: • If there is a new Allocation (received for this decision logic cycle) • What was its last decision, being the last ControlSpace it did submit • What is the history of decisions (previous ControlSpaces submitted and previous Allocations received and processed)
Energy Application Layer	EnergyApplication Logic Manager (ELM)	Manage the decision logic cycles of its decision logic components, being the EnergyApplication
	EnergyApplication World View	Provide a correct view for the EnergyApplication on all information it needs to make a decision. It needs to know for example: ▪ If the are new EnergyAssignments (received that it did not consider before in previous decision cycles) ▪ If there are new ControlSpaces (received that it did not consider before in previous decision cycles) ▪ What was its last decision; a set[39] of EnergyProposals submitted, what were its last decisions; Allocations, submitted to ResourceManagers. ▪ What is the history of decisions submitted (set of EnergyProposals and Allocations) and received (EnergyAssignments and ControlSpaces). ▪ The current values and history of total consumption and generation information.

Besides the WorldViews, decision logic components can have additional information available to take into account in their decision-making. That information can be provided by Helper Apps, which either produce information on their own or make information available from ThirdPartyInformationProviders.

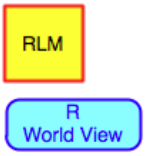
In the 3rd step, the ECS uses its decision logic managers to trigger a new decision logic cycle in the decision components of the layers. Via the F1/F2 external communication interfaces decisions flow between primary and secondary nodes. It is not a design requirement that the ECS of the primary and secondary nodes perform an orchestrated 3rd step. Since the ECS in-queue of the primary node provides a “landing strip” for decisions both from the primary and secondary decision components, the decision logic cycles do not need to be executed in an orchestrated way.

The decision logic managers play an important role in ensuring a correct and consistent environment during a decision logic cycle.

[38] Note that the R WorldView provides a separate view for each ResourceManager, since Resources do not and should not know about each other.
[39] Note that an EnergyApplication should submit a consistent set of EnergyProposals.

- They provide a correct environment by making sure that the WorldView has outdated[40] information marked, so that the decision logic component can take this into account.
- They provide a consistent environment in the following way. Functionally speaking, they set a marker on the mailbox of the layer and trigger the decision cycles of the decision logic components in the layer, so that they can execute their decision logic taking into account all past and new information from the mailbox up to the marker. Other realizations are possible, such as halting the pick-up of items from the in-queue, halting the router, finalizing the delivery to all mailboxes before triggering the decision cycles.

We hereby introduced the following subcomponents of the ECS responsible for providing ECS guidance in/for the layers of the Node:

Symbol	Subcomponent	Responsibility
	EnergyApplication Logic Manager and EnergyApplication World View	ELM triggers decision logic cycle of decision logic components of the EnergyApplicationLayer, provides a correct and consistent view during decision logic cycle. The E World View provides the decision logic components the view on the information they need for their decision logic.
	Resource Logic Manager and Resource World View	RLM triggers decision logic cycle of decision logic components of the Resource Abstraction Layer, provides a correct and consistent view during decision logic cycle. The N World View provides the decision logic components the view on the information they need for their decision logic.

4.2.2.5.4.6 The structure of an energy "message"

An EDM is a representation of an energy-related decision or piece of information that is important for energy management. It is the registration of an event of submitting, routing or receiving decision information. An EDM has the following attributes:

Attribute	Meaning	Comments
Source	Identification of the original decider/sender	This is a specific identification, referencing a specific instance of a version of a ResourceManager for example.
Destination	This is an identification of the destination. It is a generic destination type for submitted EDM's, becomes a specific type for routed EDM's.	A ControlSpace submitted by a ResourceManager has on submission a generic destination (e.g. "EnergyApp"). On routing a new EDM will have specific destination identification, e.g. referencing a specific instance of a version of an EnergyApplication.
Timestamp	This is the absolute-time-stamp of the EDM assigned by the SMF.	The SMF assigns timestamps at the processing step (1,2,3) according to the synchronized time-clock of the Node. For step-1, the Factory assigns the timestamp; for step-2 the Router assigns the timestamp, for step-3 the *LM assigns the timestamp.
ExpirationTime	This is an absolute-time that the decision-maker/submitter can assign, by which a response is expected back.	See important notes below on dealing with expiration times

[40] When for example a device is disconnected, ControlSpaces of its ResourceManager and maybe further decisions involving these ControlSpace (such as Energy Proposals, EnergyAssignments and Allocations) are "invalid" or should no longer be sought or be waited for. The degree in which decisions are valid is marked and available in the WorldView of a decision component. The policy of actively retracting offers based on invalid underlying decisions is not enforced by the system, but behavior that the decision logic components themselves can implement.

RoleLevel	This is an indication what security/authorization right/role is needed to process the message	
DecisionObject (reference)	This is a reference to the real DecisionObject	DecisionObjects are kept by the SMF in the WorldModel. Others have a view on this information. An EMD has a reference to a decision object. Decision Objects can be - Control Space, Allocation, EnergyProposal, EnergyAssignment, ThirdPartyDecisionInformation

An important rule for using expiration times with decision-making, everyone has to comply with, is:

When using an expiration time on submitting an energy decision, by which one expects an answer, one must provide default behavior to exhibit when no answer is received by that expiration time.

This means for example that when a ResourceManager submits a ControlSpace with an expiration time, the ResourceManager should exhibit default behavior when no allocation is received by that expiration time. When an allocation is received, it can act accordingly, when nothing is received is should decide for itself what behavior to exhibit.

In other words, submitting decisions does not guarantee an answer back within the specified expiration time, or even ever. One should specify default behavior when no answer is received in a given expiration period, or when no answer is received at all.

The next diagram shows the different kinds of decision objects an EDM can refer to.

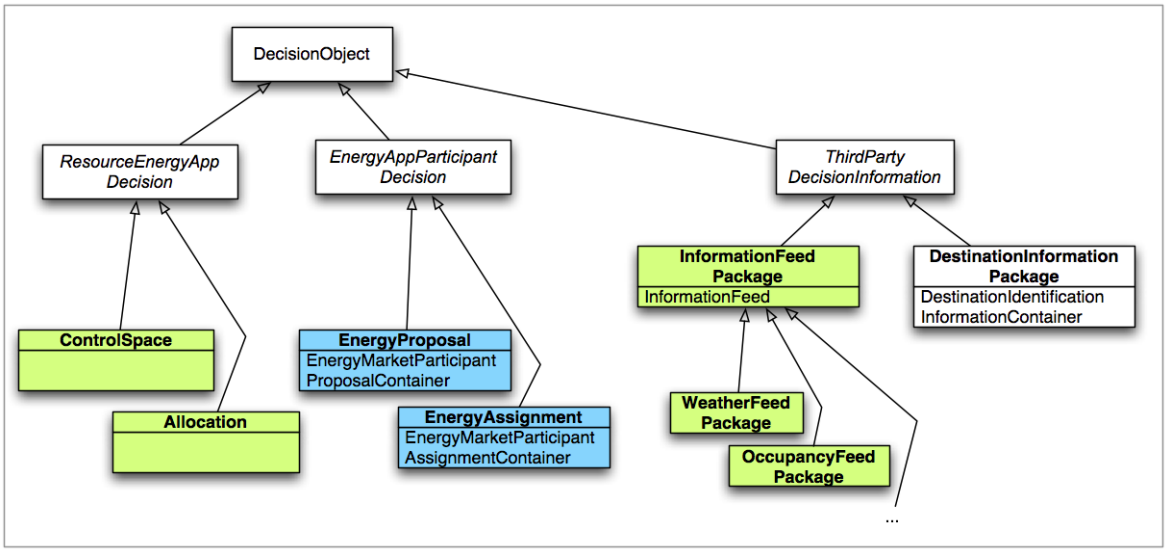


Figure 4-26 Types of decision objects and EDM can refer to

Note that ControlSpaces and Allocations are generic representations. EnergyProposals and EnergyAssignments are container objects containing a payload with a representation specific for an EnergyApplication and its EnergyMarketParticipant protocol.

Note that ThirdPartyDecisionInformation can be generic if it is information for a standardized feed. It can also be specific and only understandable for a specific destination (e.g. an App which receives third party information from its server-side). The previous figure illustrates the case of generic information with two examples; a package of information for a weather forecasting feed and

a package of information of an occupancy feed. The latter example has information about the presence/absence of the occupants of the account premises, which could be based on location information of the account members extracted from location sensor information from their smartphones. This kind of information can be valuable for energy management of home heating and lighting systems.

4.2.2.5.4.7 The WorldModel and WorldViews

We discuss important aspects of WorldViews and their relation with the WorldModel. Conceptual diagrams are presented on the information provided by the different WorldViews.

4.2.2.5.4.7.1 Separation of Decision Objects and their relations

As introduced earlier, the SMF has a SMF Model that has a WorldModel. The WorldModel contains the Decision Objects, such as ControlSpaces, etc. The WorldModel also knows about the Decision Object Relations, so its knows what ControlSpaces were considered and which were used for making an EnergyProposal, which proposal is related with an EnergyAssignment, what Allocations are related with which EnergyAssignments.

The DecisionObjects themselves do not contain this relation information, since what relation information knowledge is required depends on the party involved. A ResourceManager does not need to know about underlying ControlSpaces of an EnergyProposal when receiving an Allocation based on an EnergyAssignment for a proposal, but an EnergyApplication needs to know this. An Allocation decision object does not contain information about underlying EnergyAssignments, these relations are separately represented in the DecisionObjectRelations part of the WorldModel. The next diagram shows the structure of the WorldModel as introduced earlier.

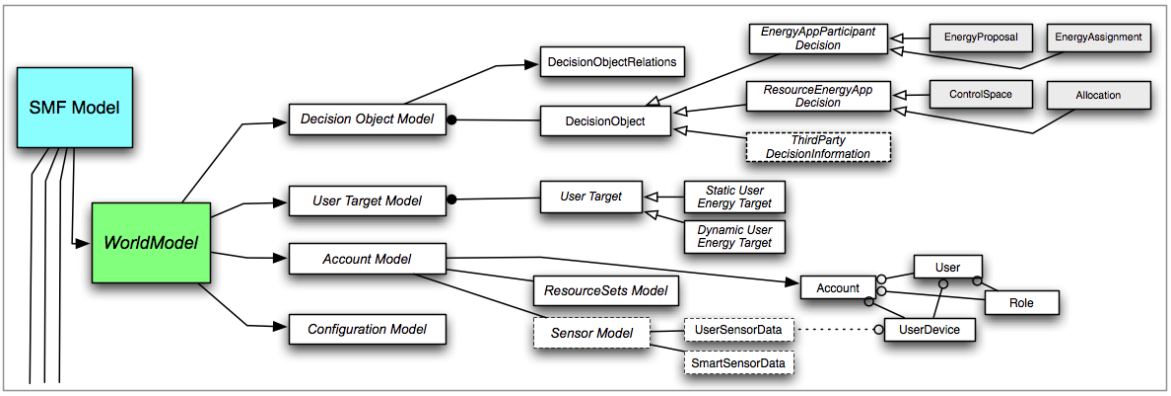


Figure 4-27 WorldModel of the SMF Model

4.2.2.5.4.7.2 Conceptual diagrams of WorldViews

We introduce for each of the WorldViews a conceptual diagram of the information provided. Note that these are conceptual diagrams that illustrate the different categories of information available in a WorldView, this does not imply that a WorldView should be technically realized as such a structure.[41]

[41] One probably should provide the layer (e.g. EnergyApplicationLayer) and its decision component (e.g. an EnergyApp) an interface on a DecisionObject that makes the relevant information visible (e.g. underlying relations with ControlSpaces, validity information, etc.).

a) Conceptual diagram of the EnergyApp WorldView

In the conceptual diagram for the EnergyApp World View, one observes Energy Assignments and ControlSpaces as “new decisions received”, together with Energy Proposals and Allocations as “last decisions submitted”. For all, the “history of decisions” is also available.

Structural information contains the relations with other decision objects to the degree that the layer is allowed to see them, together with validity information of the decision objects and the related ones. For example, the underlying relations of an EnergyProposal with ControlSpaces it is based on is visible; they are however not visible for a ResourceManager.

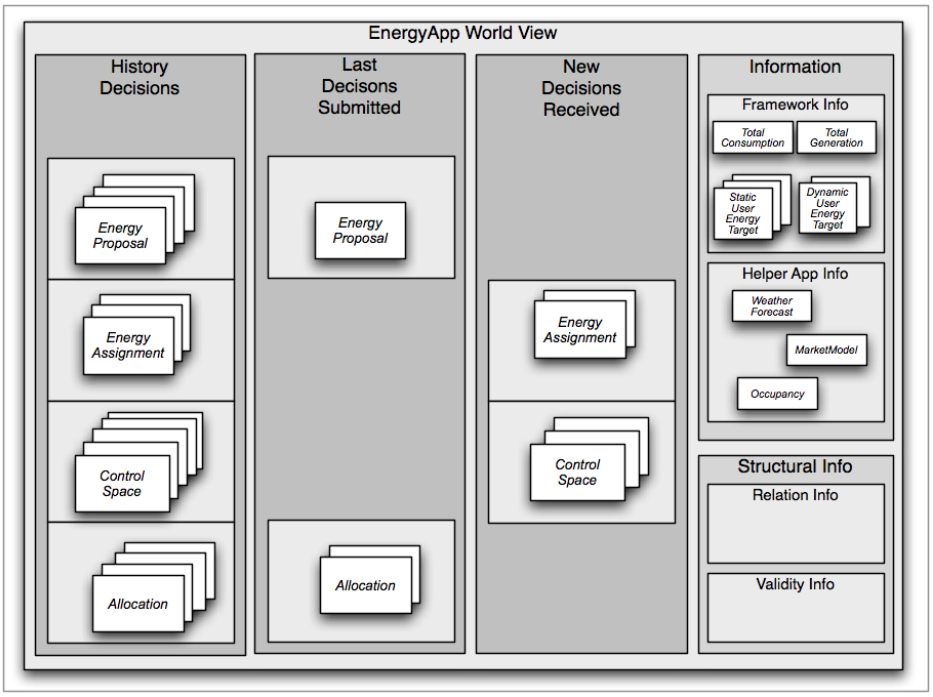


Figure 4-28 Conceptual diagram of EnergyApp WorldView

Framework information as also part of this WorldView. The total consumption and generation information is part of this. Also static energy user targets (profiles) are available in this View, as well as dynamic settings for specific resources.

Helper App information can also be part of the WorldView. The figure above shows examples of weather forecast information, a market model[42], or occupancy[43] information.

b) Conceptual diagram of the Resource WorldView

In the conceptual diagram for the Resource World View, one observes Allocations as “new decisions received”, together with ControlSpaces as “last decisions submitted”. For all, the “history of decisions” is also available. Structural information and validity information is similar as before. As part of the Framework Information one observes the static user energy targets since

[42] An App could provide an estimate market model so that it provides insight into what it can expect when offering proposals to a market participant, or give the EnergyApp a broader sight on the entire market behavior.

[43] An App could provide occupancy information of the account premises by using location information from smart phones of account users, or use information from a GUI calendar app where account user indicate their work schedule and holiday periods, or use information from flipping a switch installed in the living room next to the front door for users to indicate when the leave and enter the house. An example of the latter is the Intel® Intelligent Home Energy Management Proof of Concept.

they belong to the ResourceManager for which the Resource World View is offered. Note in the diagram possible helper app information, such as a weather forecast from a Weather App helping the Resource Manager. Another example shown in the figure is outside temperature, which could be provided by an App that provides this information and its expectations, maybe based on temperature readings it receives and collects from a smart meter for ambient conditions such as room and outside temperature.

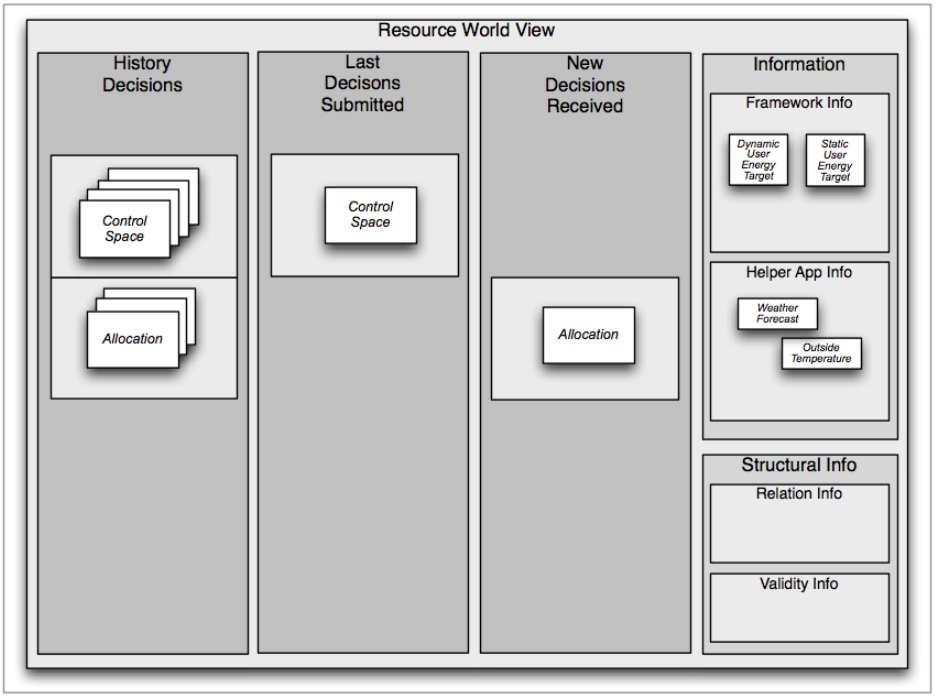


Figure 4-29 Conceptual diagram of Resource WorldView

4.2.2.5.4.7.3 Functional versus technical realization

In the previous paragraphs we introduced the SMF Model as a SMF object repository containing the WorldModel, EventLog, CoreStorage and SMModel. The WorldModel contains Decision Objects and separately their relations. We also specified WorldViews for the decision logic components of the layers to have all information available for executing their decision cycles. A DecisionLogicManager was defined to trigger the decision logic cycles and to provide a correct and consistent WorldView.

These are all functional artifacts that do not necessarily have technical equivalents. For example, a WorldView can be realized by providing interfaces for DecisionObjects that look for the respective decision components as DecisionObjects but in fact provide an aggregated view realized by references to the SMF Model objects.

One could for example have an Allocation interface available for an EnergyApplication, which allows the EnergyApplication to ask for the underlying EnergyAssignments, the EnergyProposals these were assignments for and the ControlSpaces the proposals were based on. The interface therefore would aggregate information from the DecisionObjects and the DecisionObjectRelations in the WorldModel of the SMF Model. Also the validity information can seem to be attributes but is in fact aggregated information.

Regardless of the chosen technical realization, the following principles should be realized:

- A **separation** between the decision objects and their relations with other decision objects

- A **difference** between SMF managed objects (such as decision objects) and objects locally managed by a decision logic component (such as an Energy Application).
- Locally managed representations and state transition artifacts (such as EDM's) should have **no direct references** to SMF managed objects but abstract references. This since the SMF must have the ability to sync and remove older decision objects, while locally managed representations and state transition artifacts are allowed to keep abstract references.
- **No direct coupling** between Energy Apps and RAL Apps. To allow for framework monitoring, communication across FP Nodes (primary/secondary nodes), dynamic configuration changes and independence of execution paths, one should realize a decoupled communication.

The diagram on the right illustrates a possible approach to realize a decoupled communication between an Energy App (the controller in the diagram) and a RAL App (the resource manager in the diagram).

As one can see, a service in between supports both push (via submission and notifications) and pull (via expressing interest triggering a request) styles of communication.

This example approach also serves as an indication that the requirements do not imply the need for a full-fledged messaging infrastructure.

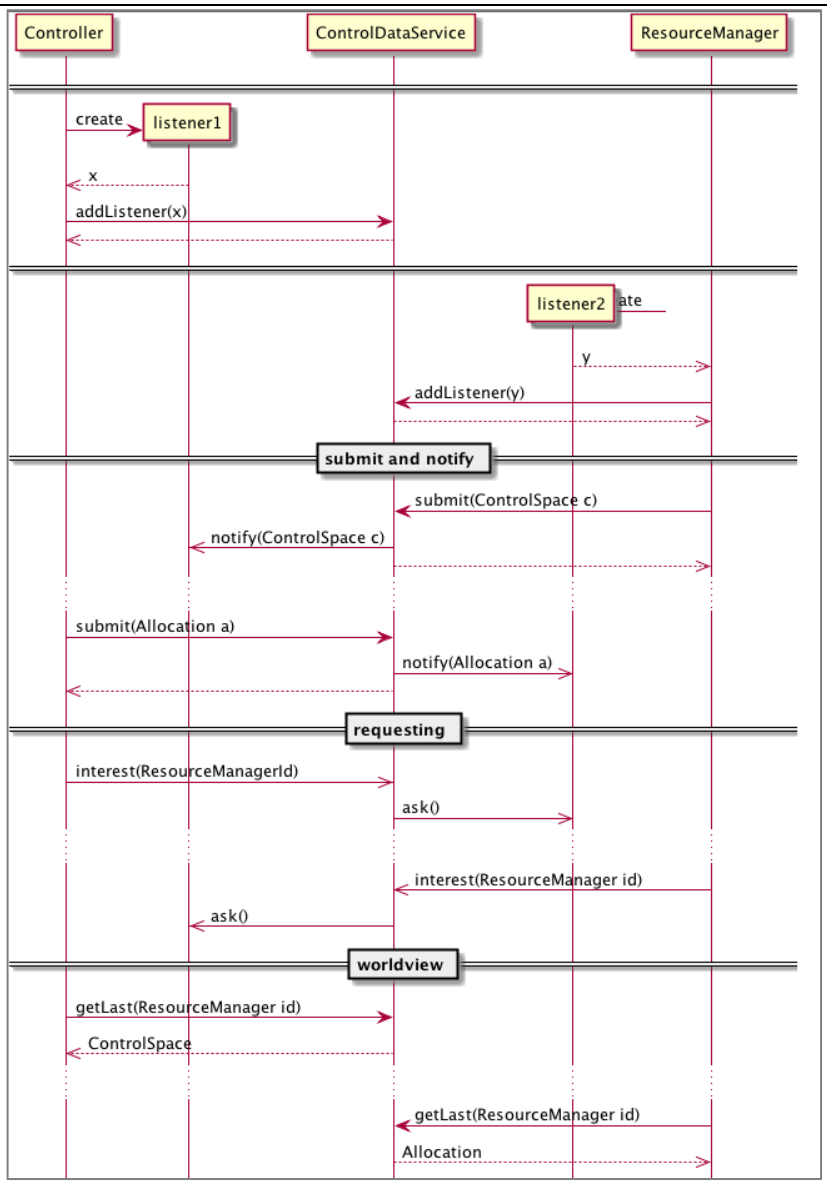


Figure 4-30 Example realization of decoupling EAL and RAL

4.2.2.5.4.8 *An example of ECS and a MessageLog with EDM's*

To clarify the ECS and its use of EDM's we give a small (functional) example of a series of ECS events and the corresponding EDM's which are created and stored in the MessageLog.
Suppose a certain ResourceManager <RM-instance 234331> with abstract reference <RM #12> makes a decision

```
ControlSpace cs = makeControlSpace(…)
```

This uses a decision object factory of the SMF to create a ControlSpace instance <CS-instance 876754> which becomes part of the WorldModel of the SMF Model; cs really is an abstract reference <CS #7>. The WorldModel of the SMF Model further knows about the relation between the ControlSpace and the ResourceManager.
When the ResourceManager submits the decision, with

```
submitControlSpace(cs,e1)
```

the ECS in its 1st step uses its Factory to create an EDM, logs it in the MessageLog (part of the SMF Model) and puts it on its In-queue.
The EDM looks like

```
[ecs:submit src: <RM #12> dest: <EnergyApp> tstamp: <t1> texp: <e1> dec: <CS #7>]
```

Note that we make abstraction in this example of the RoleLevel attribute of an EMD.
The destination is a generic destination, being "an EnergyApp", since it is not known at this point which specific EnergyApp the EMD should be delivered to.
Note that <t1> is the timestamp of the 1st step assigned by the Factory.

In the 2nd step, the ECS picks up EDM's from its in-queue and asks its Router to perform routing. The Router consults the ConfigurationModel of the SMF Model and determines that it can resolve the generic destination <EnergyApp> with a specific destination <EA #34>, being the abstract reference for the EnergyApplication currently deployed in the EnergyApplication Layer that is known to work for that resource; this is known by the ResourceSet of the EnergyApp. The Router uses the Factory to create a new EDM, adds it to the MessageLog and adds it to the out-queue.
This new EDM looks like

```
[ecs:rout src: <RM #12> dest: <EA #34> tstamp: <t2> texp: <e1> dec: <CS #7>]
```

Note that we have a new timestamp <t2> and a specific destination <C #34>.
The expiration time is the original expiration time; as it was set by the ResourceManager when submitting the ControlSpace.
The out-queue (or technically probably a thread picking up items from the out-queue) delivers the EDM to the mailbox of the EnergyApplicationLayer (or technically the EnergyApplicationLayer mailbox gets the EDM from the out-queue via a subscription implementation).

In the 3rd step, when the ECS wants to trigger decision logic cycles for decision components of the EnergyApplicationLayer, it asks it ELM to manage a next decision logic cycle for the decision components. The ELM closes the mailbox (or puts a marker on it) to provide a consistent environment, it consults the SMF to check the validity and sequences over all decision logic components of the EnergyApplicationLayer, in this case only the single EnergyApplication (with abstract reference <EA# 34>. As a side note, when the ELM would observe that <EA #34> is no longer present, for example because it has been replaced by a new EnergyApplication, it can throw the EDM back onto the in-queue, where it will later be picked up again and routed to the new EnergyApplication. This could also be the case when the ResourceSet of the EnergyApplication was changed by the User; so that now another EnergyApplication is known to be managing the resource. If the ELM can deliver the EDM, it will create a new EDM (using the Factory) to represent this fact. The new EDM looks like

```
[ecs:receive src: <RM #12> dest: <EA #34> tstamp: <t3> texp: <e1> dec: <CS #7>]
```

The ELM does this for all EMD that have <EA #34> as their destination (could be additional ControlSpaces or Energy Assignments), so that all of these receipts are known. Then it triggers the EnergyApp <EA #34> to perform its decision logic cycle. In the event that it makes and submits decisions, new EDM's are created and so on.

This gives an idea of the EDM structures and the MessageLog which builds up like:

```
[ecs:submit src: <RM #12> dest: <EnergyApp> tstamp: <t1> texp: <e1> dec: <CS #7>]
[ecs:rout src: <RM #12> dest: <EA #34> tstamp: <t2> texp: <e1> dec: <CS #7>]
[ecs:receive src: <RM #12> dest: <EA #34> tstamp: <t3> texp: <e1> dec: <CS #7>]

[ecs:submit src: <RM #17> dest: <EnergyApp> tstamp: <t1'> texp: <e2> dec: <CS #9>]
[ecs:rout src: <RM #17> dest: <EA #34> tstamp: <t2'> texp: <e2> dec: <CS #9>]
[ecs:receive src: <RM #17> dest: <EA #34> tstamp: <t3'> texp: <e2> dec: <CS #9>]

[ecs:submit src: <EA #34> dest: <EMP> tstamp: <t4> texp: <e3> dec: <EP #17>]
[ecs:rout src: <EA #34> dest: <EMP #4> tstamp: <t5> texp: <e3> dec: <EP #17>]
[ecs:receive src: <EA #34> dest: <EMP #4> tstamp: <t6> texp: <e3> dec: <EP #17>]
...
```

where in the last EDM's shown the Energy Communication Manager acts at Energy Market Participant <EMP #4> receiving an EnergyProposal <EP #17> from the EnergyApplication; in fact working with the external communication interface to send out the proposal package.

The decision logic components do not see EDM's; when they are triggered to perform a decision logic cycle they can observe everything they need from the WorldView, which gives them all information about new arrived decisions, past received and submitted decisions, structural information revealing relations and validity, additional framework information and helper app information.
Note further that Helper Apps providing information can either submit their information the same way, or rely on a ThirdPartyInformationProvider (a subscription to a general feed or their own app server part sending information) sending information, which also arrives via the ECS as EDM's. Recall that in the EDM class hierarchy there is a ThirdPartyDecisionInformation class to represent these information objects. These can either be a standardized open format information packages (InformationFeedPackage) or closed envelope private-format information packages (DestinationInformationPackage).

4.2.2.6 *Interaction Communication*

4.2.2.6.1 *High-level interaction communication flow*

The “interaction” communication stream is about the interactions with Users via UserDevices consuming a GUI, visiting the FP AppStore, using WebServices either in general or to provide a GUI on a UserDevice, or providing sensor data.

4.2.2.6.2 *Interaction Communication Scheme (ICS)*

The next figure illustrates the Interaction Communication Scheme (ICS).

The INTERACTIONFRAMEWORK can function independently from the SMF. It acquires a token-based authorization from the SMF to handle its own communication with the external world. It uses IGs to provide the GUI, IWs to provide WebServices and ISs to use sensor data. It allows visits to the FP AppStore via ASc.

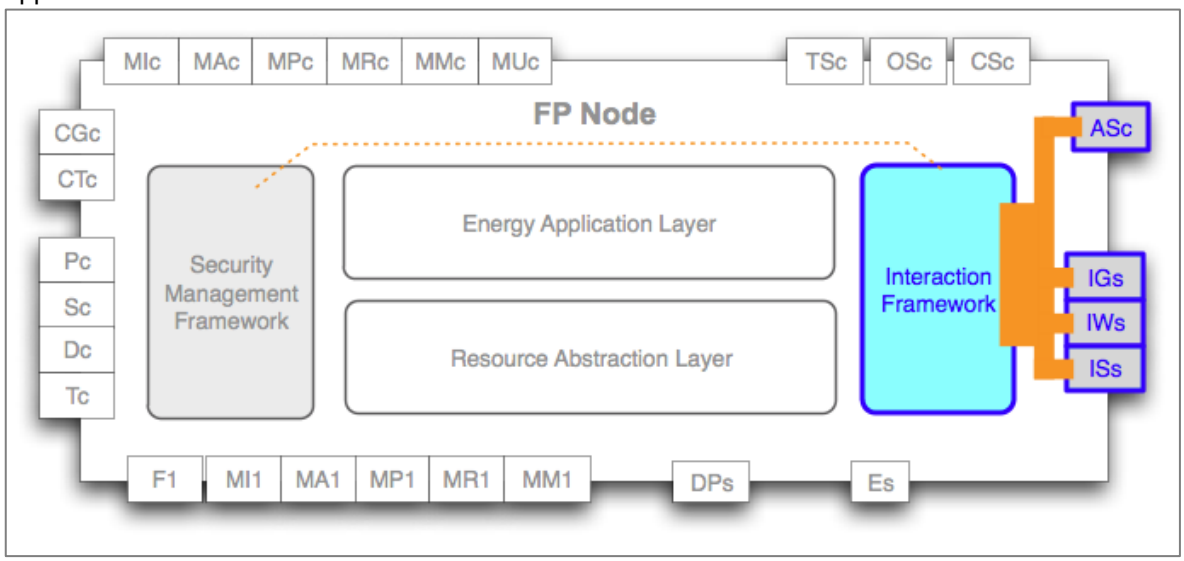


Figure 4-31 Interaction Communication Scheme (ICS) of a primary Node

Note that it is not a requirement that the ICS has a messaging nature. There is no need to represent and store messages, as was the case for the energy communication.

4.2.2.7 *Management Communication*

4.2.2.7.1 *High-level management communication flow*

The management communication involves all management related functions, including communications with the management center, communication about certificates in general, communication with other partners and management communication forwarding with secondary nodes.

The SMF manages the external communication interfaces by reading the messages received and posting answers.

4.2.2.7.2 *Management Communication Scheme (MCS)*

The Management Communication Scheme (MCS) is shown in the next figure.

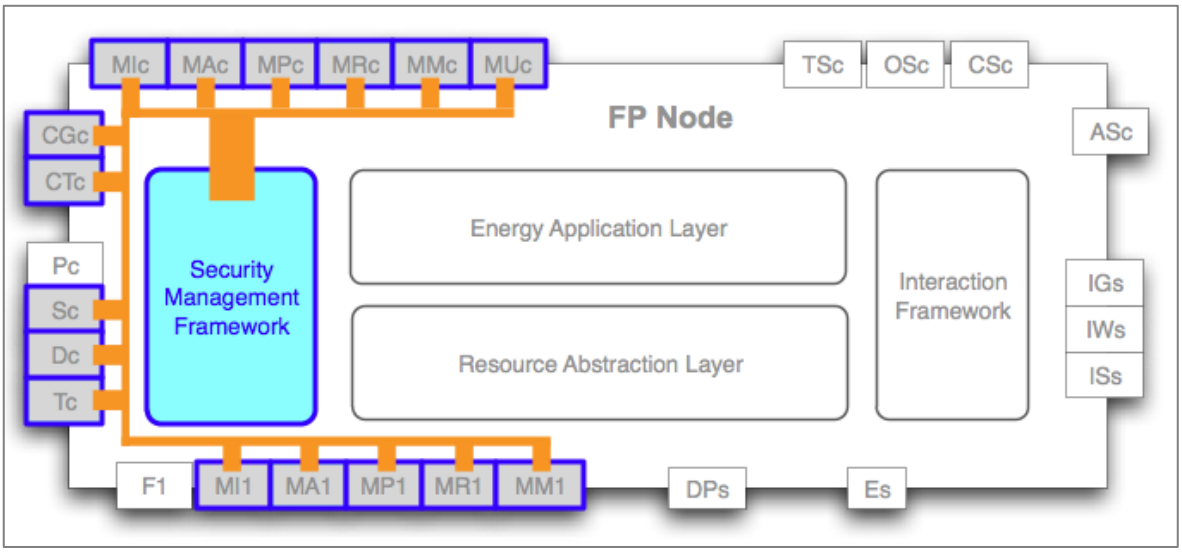


Figure 4-32 Management Communication Scheme (MCS) of a primary Node

Note that it is not a requirement that the MCS has a messaging nature. There is no need to represent and store messages, as was the case for the energy communication.

4.2.2.8 *System level communication*

4.2.2.8.1 *High-level system communication flow*

System level communication is about time synchronization, open services (installations, updates of software, bundles, plugins, apps) and configuration services (service activation & reconfiguration, remote subscriber support, firmware & configuration management, diagnostics & monitoring).

4.2.2.8.2 *System Communication Scheme (SCS)*

The System Communication Scheme (SCS) interacts with the system via the external interfaces TSc, OSc and CSc.

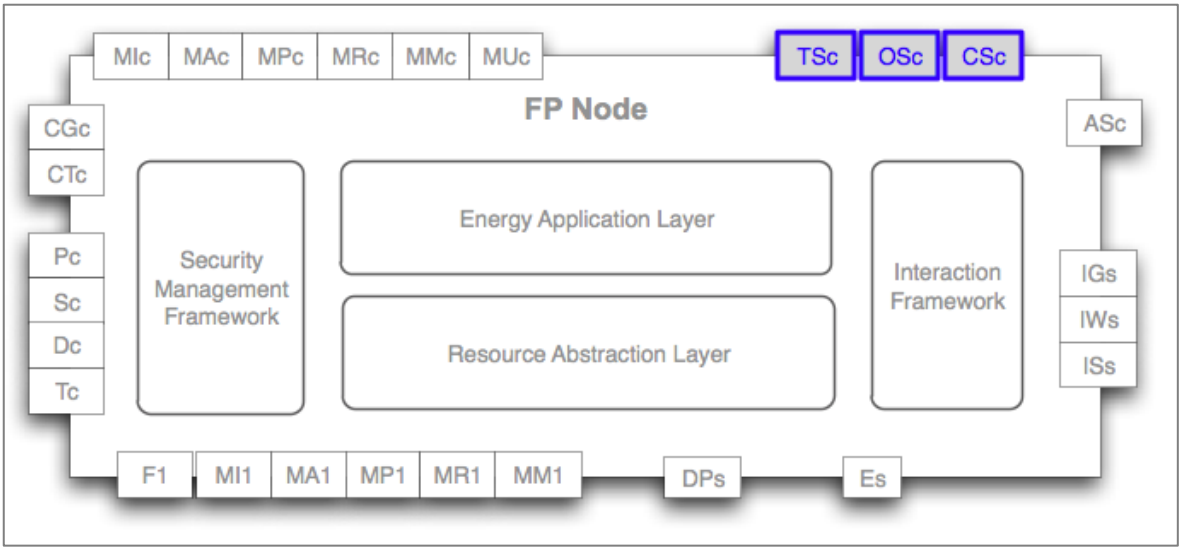


Figure 4-33 System Communication Scheme (SCS) of a primary Node.

Note that it is not a requirement that the SCS has a messaging nature. There is no need to represent and store messages, as was the case for the energy communication.

4.2.2.9 *Types of Resources and related decision objects*

4.2.2.9.1 *ResourceManagers*

Resources represent physical consumption/generation devices in the FP Node. They have a ResourceManager that abstracts the device into a time-shifter, buffer, storage or uncontrolled load/generation device. This ontology of resource managers can be more refined by specializing the four top-level categories, but at least has these top categories.

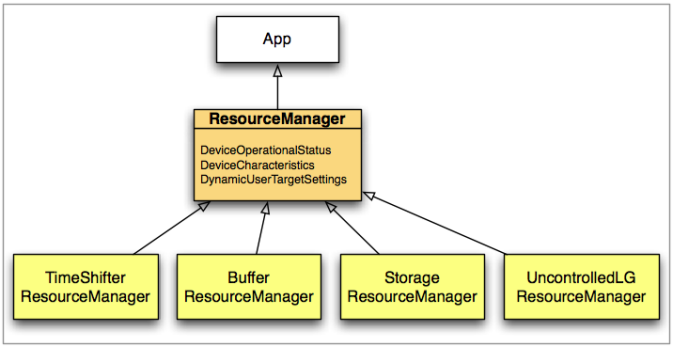


Figure 4-34 Ontology of ResourceManagers reflects abstract device types

A ResourceManager knows about

- The operational status of the device. It receives operational status information via the ResourceDriver communicating with the device.
- The characteristics of the device. This knowledge is encoded in the specific ResourceManager App implementation.
- The dynamic user energy target settings for the device, which are possible ad-hoc deviations from the default static user energy target settings.

Furthermore, a ResourceManager has access to additional information from its WorldView as already discussed. This includes the default user energy target settings for the device and additional information provided by its Resource Manager Helper Apps.

4.2.2.9.2 *ControlSpaces*

The ControlSpace produced by the ResourceManager exposes the energetic needs/provisions and its flexibility; the specifics are condensed into four abstractions corresponding the four categories of Devices and top-level ResourceManager types.

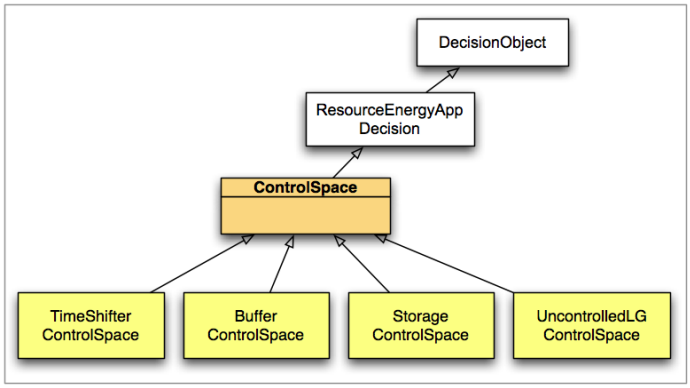
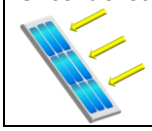


Figure 4-35 ControlSpaces, reflecting the ontology of ResourceManagers

Although the ontology of resource managers can be more refined to specialize the top categories, the ontology of control spaces is limited to the four top-level categories. These serve as abstractions for the EnergyApp decision logic that must be independent from further specialization of Resource Managers.

4.2.2.9.3 *Categories of ResourceManagers and their ControlSpace parameters*

Devices are abstracted in four categories. These correspond with the four top-level types of Resource Managers as well as with the four types of ControlSpaces. The next table describes briefly the different categories of Devices, gives some examples and summarizes the main parameters of their ControlSpace.

Category	Description	Examples	ControlSpace (main) parameters
TimeShifter 	TimeShifters consume/generate energy according to a predetermined energy profile, the start time of this profile can be shifted in time over a given period.	washing machine or dish washer that allows shifting the start moment	Energy profile (Array of Wh) and period for start moment of profile (StartAfter, StartBefore).
Buffer 	Buffers can consume/generate more or less energy (within certain operational constraints) according to the needs of the account. They could temporarily consume / generate more energy so that later they need less/more.	Thermic buffers like household heating systems, freezers and refrigerators. Note that micro-CHP systems temporarily generate more.	Total buffer capacity (Wh), State of Charge (%), Charge speed/curve (W), Discharge speed/curve (W), minimal switch on period, minimal switch off period.
Storage 	Storage devices can store electricity and release it when required. They are similar to a buffer but can both take and return energy.	batteries, electrical vehicles	Same as buffer, but also parameters for *energy turnover losses (losses in conversion)*
UncontrolledLG 	The Uncontrolled Load/Generation category are devices for which the energy behavior cannot be actively controlled. One could make forecasts.	PV (Photovoltaic) panels, television sets, computers, lighting.	Forecast of the expected consumption/generation (predicted energy profile, as array of Wh)

4.2.2.9.4 *ControlSpaces parameters*

The following diagram shows the main parameters of the ControlSpaces.

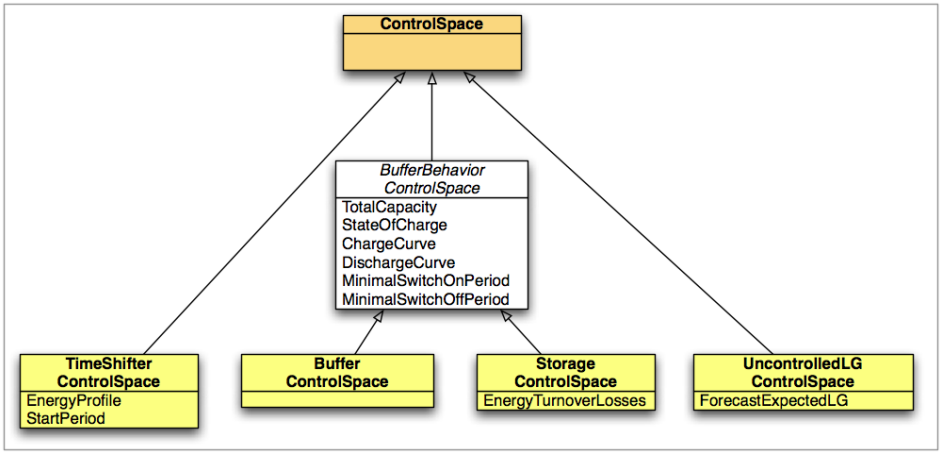


Figure 4-36 ControlSpaces with parameters

An EnergyProfile is typically expressed as an Array of Wh values. A StartPeriod has a StartAfter and StartBefore time point. Buffers and Storage types have the same parameters, except for additional Energy Turnover Losses (conversion losses) parameters of a Storage type. UncontrolledLG types have a forecast of expected consumption and generation.

4.2.2.9.5 *Allocations*

Devices, abstracted as Resources with a ResourceManager and ResourceDriver, expose their energetic needs and flexibility as ControlSpaces.
The coordination process of the EnergyApp results in energy assignments received from the energy market, for which the EnergyApp examines what ControlSpaces they cover. Such coverage is communicated back to the Resource as an Allocation. When the ResourceManager does not receive an Allocation (in time), it will proceed with a default behavior. However when it receives an Allocation, it (could) control the Device to comply with the Allocation.

Note that not all Allocations made by an EnergyApp need to be based on energy assignments received from an EnergyMarketParticipant. An EnergyApp could also make allocations based on the energetic flexibility of its consumption and generation devices to achieve a balance within the account/household provisions.

The Allocation types correspond to the four categories of Devices and top-level ResourceManager types, as illustrated in the diagram below.

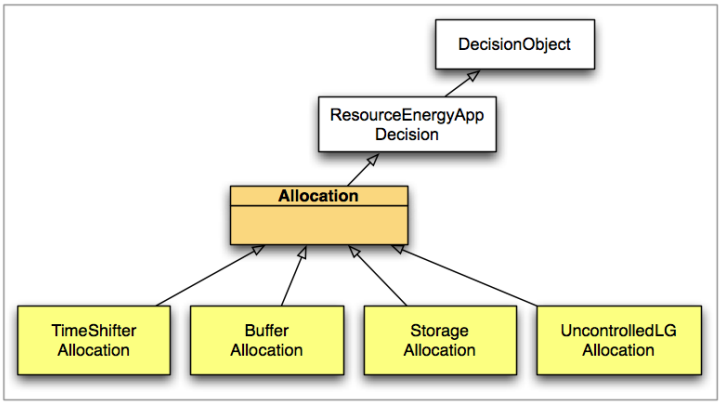


Figure 4-37 Allocations, reflecting the ontology of Resource Managers

The structure of Allocations will be further specified in iterative development/design cycles.

4.2.2.9.6 *EnergyProposals and EnergyAssignments*

An EnergyApplication can create an EnergyProposal or a set of EnergyProposals based on the energetic flexibility information from the ControlSpaces and its overall knowledge about user energy target settings, total consumption and generation information. It also can use additional knowledge provided by EnergyApplicationHelper Apps and information provided by ThirdPartyInformationProviders, either to itself or to the helper apps.
An EnergyProposal is communicated to an EnergyMarketParticipant, which can respond with an EnergyAssignment. The EnergyApplication can decide on Allocations for the ControlSpaces covered in an EnergyAssignment and communicate these Allocations to the Resources.

Remark that EnergyProposals and EnergyAssignments are container objects that have a specific payload in the format of the specific EnergyMarketParticipant for which the EnergyApplication is equipped/designed. As described earlier, the current FP Application Infrastructure platform design does not offer an separate abstraction layer for EnergyMarketParticipants or a generic way of expressing proposals or assignments.

This will be further specified in iterative development/design cycles.

4.2.2.10 Composition of Layers and Frameworks

4.2.2.10.1 Overview

The composition of an FP Node can be represented by the next figure.

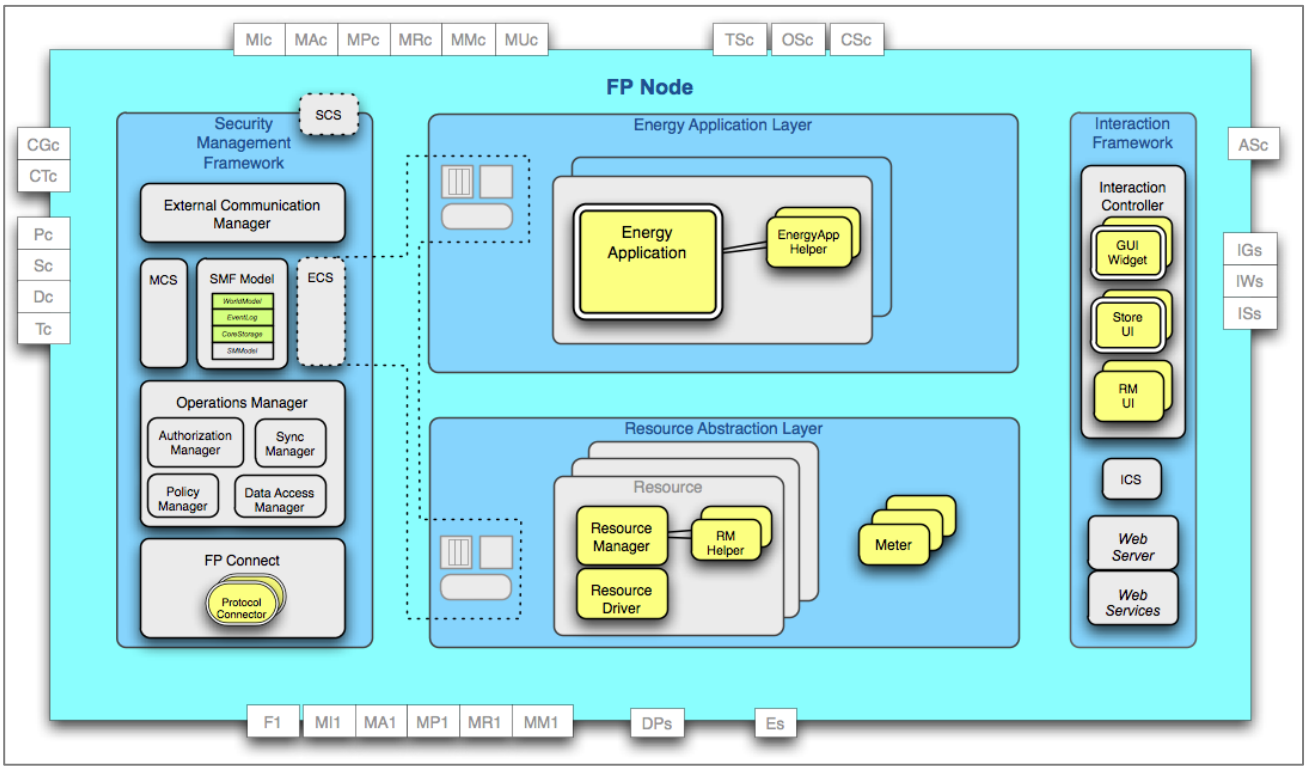


Figure 4-38 FP Node decomposition

The figure shows

- The external communication interfaces at the border
- The Security Management Framework on the left, the InteractionFramework on the right.
- The decision layers in the middle; the EnergyApplicationLayer and the ResourceAbstractionLayer
- The main subcomponents of the frameworks and layers. Note that the provisions of the ECS of the Security Management Framework stretch out with provisions for each of the decision layers.
- The figure shows in yellow all places for replaceable components.
    - Apps are yellow boxes with single-line border
    - Plugins (which can be replaced by Apps, or for which one can install additional Apps) are yellow boxes with double-line border.
    - The Protocol Connector components (which are not necessarily bundles) are shown in yellow ovals with double-line border.

In the next paragraphs we survey the aspects from this composition.

4.2.2.11 *Composition of the Security Management Framework.*

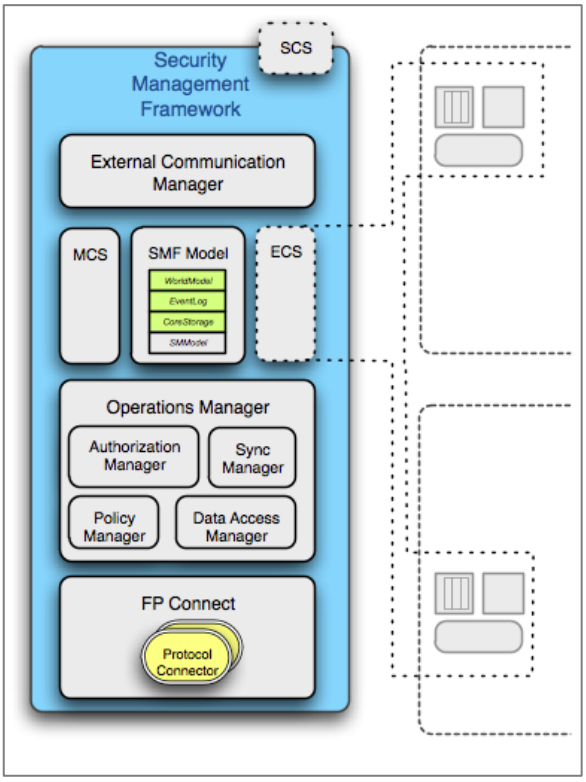


Figure 4-39 Composition of the SMF

4.2.2.11.1 *The SMF Model*

The main "state" of the Node is kept in the SMF Model. Its WorldModel, EventLog and CoreStorage parts are synced periodically with the FP ManagementCenter. The Sync Manager of the Operations Manager will manage this process. The tracking information of this sync process is also kept at the SMModel part of the SMF Model.

All SMF components are involved with the "state" of the Node, they therefore all have an interaction with the SMF Model. The next diagram shows the parts of the SMF Model and the main interactions of the SMF components with those parts. In the review of the SMF components in the next paragraphs these interactions are described in more detail.

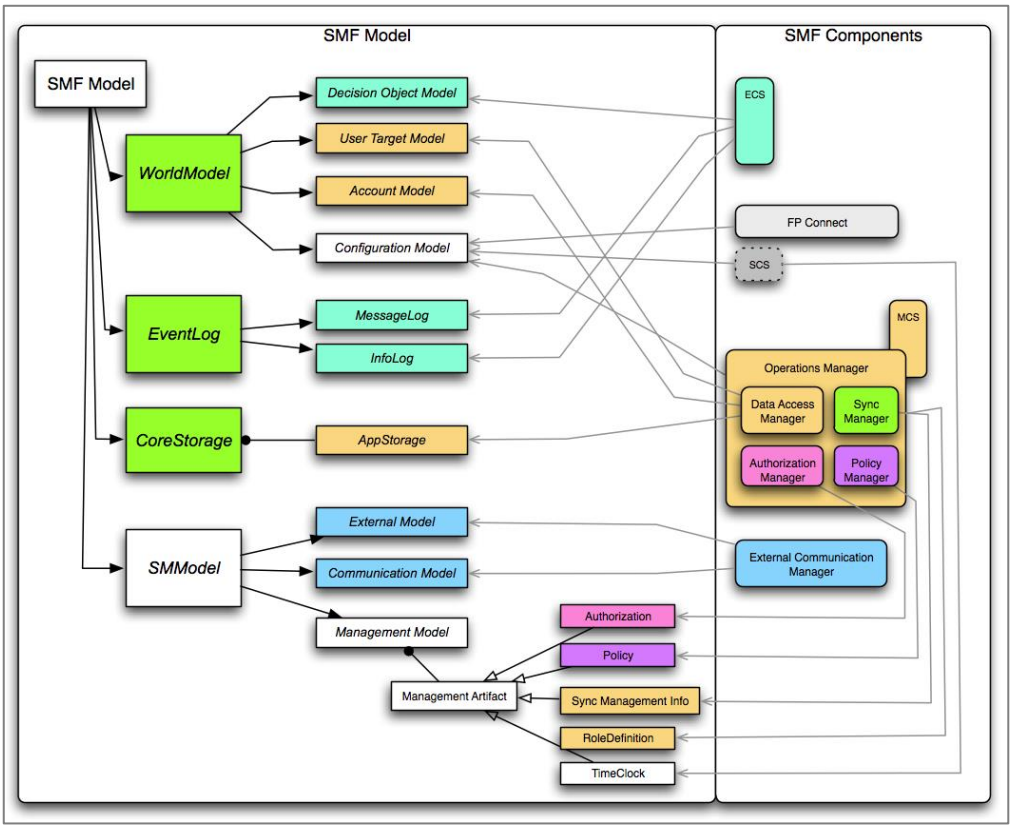


Figure 4-40 Interactions of SMF components with the SMF Model parts

4.2.2.11.2 External Communication Manager, ECM and ECS

The External Communication Manager (ECM) interacts with those external communication interfaces of the node that are about energy or management related communication levels. The communication schemes involved are the MCS and ECS.

Note that the SCS is not fully part of the SMF, but more situated on a system level.

The ECM uses the Communication Model part of the SMF Model to store and access communication artifacts (security policies, certificates, encryption keys, etc.). It uses the External Model part of the SMF Model to provide communication details to the external communication interfaces.

The ECS provides internal communication between decision components and their energy related information interactions with external parties. The dotted lines in the SMF composition diagram indicate that the ECS provisions are also situated in the decision layers. These provisions were already discussed and include artifacts such as a MailBox, a Decision Logic Manager and the WorldViews.

The MCS provides the overall management of communications with the management center, communication on certificates in general, communication with other partners and management communication forwarding with secondary nodes.

4.2.2.11.3 FP Connect

The FP CONNECT component contains the Protocol Connectors that link the protocol driver components/functions of the Device Driver Apps with the devices via the DeviceProtocols.

FPConnect uses the Configuration Model from the SMF Model to register protocol information when protocol connectors are updated/added.

Note that the FPConnect components of the SMF realizes the **FP Connect** specification of the FP Application Infrastructure; therefore it covers all of the following:

- Communication with local devices,
- Communication with remote devices via forwarding services with secondary FP Nodes
- Communication with embedded resources to exchange control spaces and allocations with the resource manager embedded in an appliance.
- Communication for information readouts from smart meters; such as overall consumption/generation meters, smart meters for appliances or appliance groups, ambient sensors such as temperature sensors, and other smart meters
- Communication to support a remote FP HomeBox configurations, where the FP Node is deployed on a remote server, and communicates with the home area network of the account premise via device protocols to the appliances.

4.2.2.11.4 Operations Manager

The Operations Manager is the component responsible for managing the operational state of the Node. It has many responsibilities, corresponding with its subcomponents:

4.2.2.11.4.1 Authorization Manager

The Authorization Manager is responsible to manage and provide authorizations for services. It uses the authorization information in the SMModel and, according to the defined Policies, requests authorization to the Management Center via the External Communication Manager.

It also manages the delegation of communication responsibilities for user related control to the InteractionFramework, in the sense that it provides a token-based authorization so that the InteractionFramework is authorized to perform external communication for control related issues independently.

4.2.2.11.4.2 Data Access Manager

The decision logic components of the Node, situated in the decision layers, have access to the energy related information via their WorldView, as described in the section on the ECS.

There are however other needs by different components in the Node to access information. This information access is managed by the Data Access Manager via a number of interfaces/services. The following are some of these information access interfaces:

- Interface for the InteractionFramework to access and change the Account Model part of the WorldModel, to view/add/change account aspects such as Users, User Devices, Roles, etc. Note that sensor data of User Devices is also part of the Account Model (see further in paragraph on metering data service).
- Interface for the InteractionFramework to perform changes to the User Target Model part of the World Model, to view/add/change static user energy targets and dynamic user energy targets.
- Interface for the EnergyApplicationLayer (in fact its WorldView) to access the static user energy targets, etc.
- Interface for the ResourceAbstractionLayer (in fact the WorldView for a ResourceManager) to view and change the dynamic user energy targets, etc.
- It provides Apps/Plugins with access to the generic storage facility CoreStorage.

To clarify the above, note that User Energy Targets come in two flavors.

- The so-called “static” user energy targets are defaults or profiles. They are entered by the User via the GUI; the InteractionFramework uses the services of the SMF Operations Manager to store/change them in the User Target Model of the World Model of the SMF Model.
- The so-called “dynamic” user energy targets are ad-hoc changes to the energy targets for a specific device. They are entered by a User; directly on the device or via the GUI. In the first case, the device communicates via a Device Protocol with the Resource Driver, which provides the target information to the ResourceManager, which uses the services of the Operations Manager to store/change them in the User Target Model. In the second case, the GUI provides them via the InteractionFramework to the Operations Manager, which stores them in the User Target Model.

The Data Access Manager can resolve data requests even if they exceed the window of information of the SMF Model of the Node. They resolve the part of an information collection request available in the SMF Model. For the part of the requested information window not available any more they forward an InformationCollectionRequest to the FP ManagementCenter via the MI external communication interface. The received collection is merged with the local information to provide the answer. Also secondary nodes can do this by forwarding requests to their primary node.

4.2.2.11.4.3 Policy Manager

The Policy Manager is responsible to manage policies and policy updates as received from the FP ManagementCenter. It uses the policy information in the SMModel part of the SMF Model and helps the Operations Manager in enforcing the policies.[44]

4.2.2.11.4.4 Sync Manager

The Sync Manager component of the Operations Manager manages and tracks the sync operations with the FP ManagementCenter (or with its primary node in case of a secondary node).

It was already outlined in the previous chapter how the SMF should manage incremental sync operations using sync points and keep track of safe (synced before), almost safe (being synced) and fresh (new since sync request) SMF Model information.

The overall method is applied to the WorldModel, EventLog and CoreStorage parts of the SMF Model.

4.2.2.11.4.5 Other responsibilities

The Operations Manager of the SMF is also responsible for managing installations and updates of Apps of the FP Node. However this functionality relies on overall OSGi services needed for the entire FP Node software. The so-called Application Manager that handles this is not shown as part of the SMF, it is in use for the overall software installations on the FP Node.

[44] As a technical realization it could provide the access to the policy information, where all policy-dependent functionality in the SMF first asks the Policy Manager for the policy parameters when they apply a policy-dependent action.

4.2.2.12 Composition of the Resource Abstraction Layer

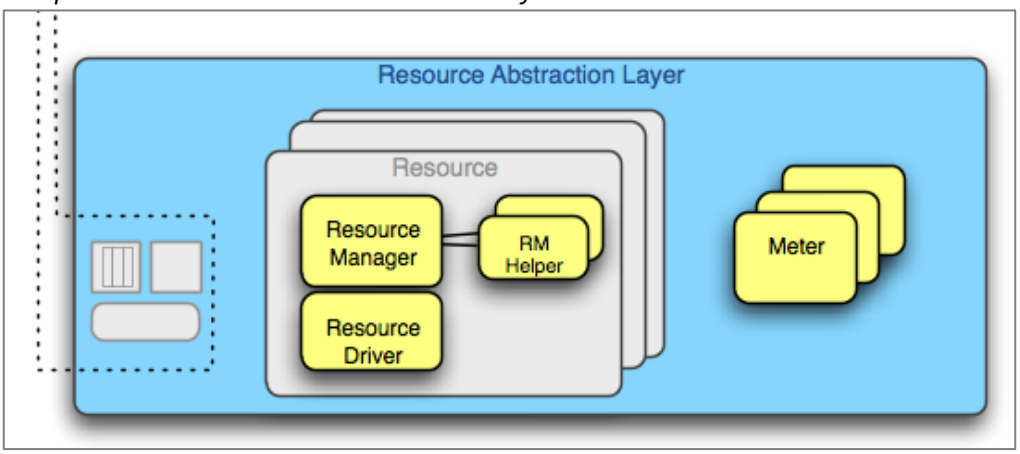


Figure 4-41 Composition of the Resource Abstraction Layer

4.2.2.12.1 ECS provisions

The Resource Abstraction Layer (RAL) as any decision layer, has ECS provisions; a MailBox, a Decision Logic Manager and the provision of a WorldView for its decision logic components.
The decision logic components of the RAL are Resources (in fact their ResourceManagers).

4.2.2.12.2 Resources

The RAL has zero or more Resources to represent physical devices monitored/controlled by the Node. A Resource consists of a ResourceManager APP and a ResourceDriver APP. Optionally it can have ResourceManager Helper APPS.
ResourceManager Helper apps could for example help with provision of weather (forecast) information.
The ResourceManager and ResourceDriver typically belong to an APPGROUP.
Note that in the paragraph about the composition of the InteractionFramework, one will see additional Apps related with ResourceManagers that can help provide a GUI page for controlling the specific device.

4.2.2.12.3 Meters and the Metering Data Service

The main abstraction of appliances/devices is done with Resources that provide energetic flexibility information via a ResourceManager App, working with a ResourceDriver App that connects to the appliance via its ProtocolDriver.
Besides energetic flexibility, metering data in general can also be of use for Apps or to visualize to the user in the GUI. This data can be about electricity consumption of an appliance, but can also be about temperature readings or other appliance or ambient information.

The sensor/metering data can be provided by a resource driver, when the appliance and the protocol used allows for reading the data. When this is not the case, or in general for appliances or groups of appliances that are not abstracted in resources, one can install Meter Apps to make the connection. In both cases, one supplies metering information about an appliance or appliance group to the data access manager, and one can access the information from there. The next diagram illustrates this, by showing both resources and meters providing metering data, and Apps accessing the information. The MDS service shown is a MeteringDataService, which is part of the general data access manager concept.

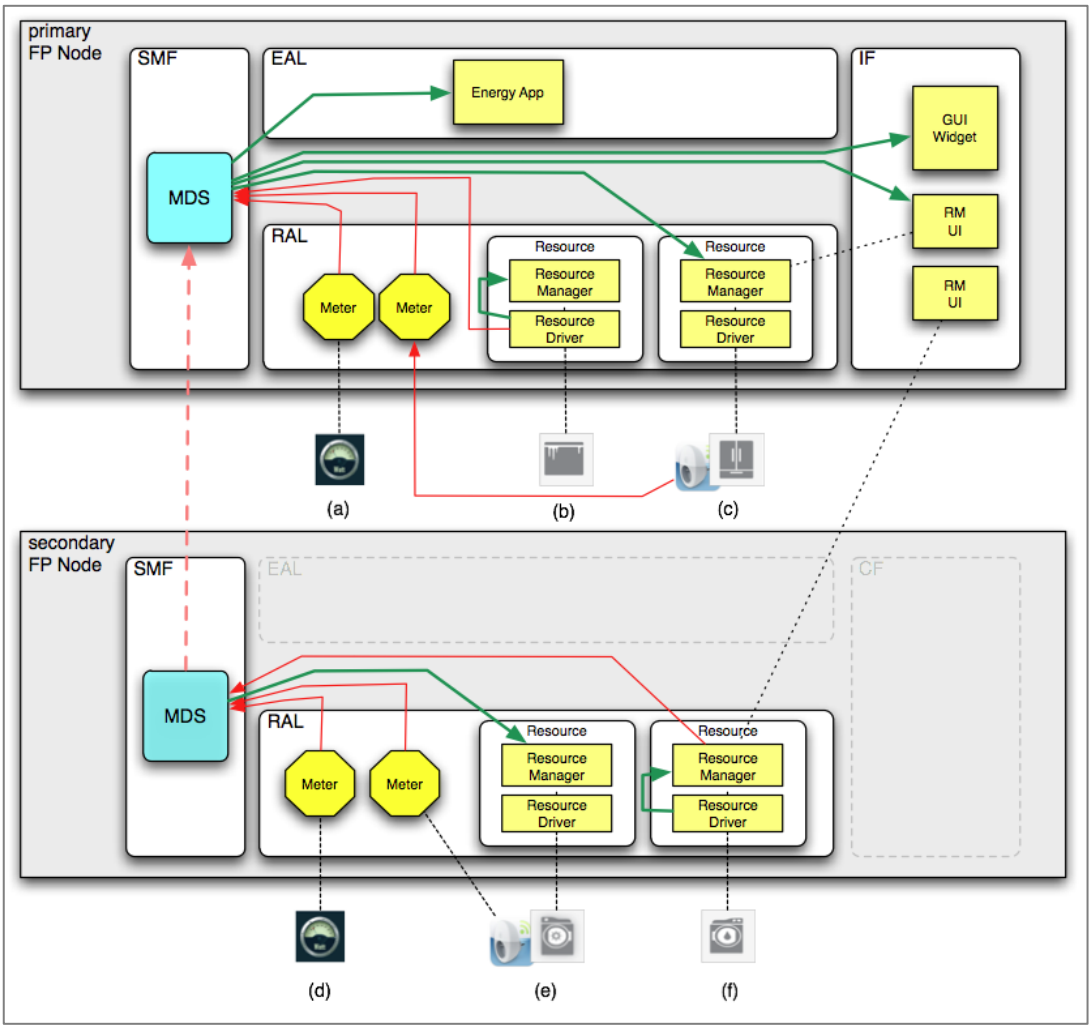


Figure 4-42 Meters and information flow

The diagram above shows a primary and secondary FP Node, where the information flow of metering data is shown with red arrows for provision and green arrows for access. With (a) and (d) we have Meters reading overall electricity consumption and generation of both locations. Appliance (b) had a resource driver and protocol supporting readout of electricity consumption; therefore its resource driver can provide the data to its resource manager as well as the MDS. For appliance (c) the resource driver and/or its protocol does not support the readout, therefore a smart-meter plug is added to the appliance with a Meter in the RAL providing the information. Note that in that case its resource manager does not obtain the data from its resource driver but from the MDS. Similar situations are shows for the secondary location where (f) had readout capabilities via its protocol and driver, where (e) needs a separate plug and Meter. For completeness the diagram illustrates that the MDS of secondary and primary nodes are linked via the general forwarding principles. The dotted lines from both RM UI Apps indicate which RM UI shows information about which appliance.

Note that the provisions of permissions and rights in the framework restrict who (which Apps) can access metering data from which appliance or group. For example a resource manager should only be able to access data about its own controlled appliance, not about other appliances.
The diagram below shows a possible technical interface for a MeteringDataService:

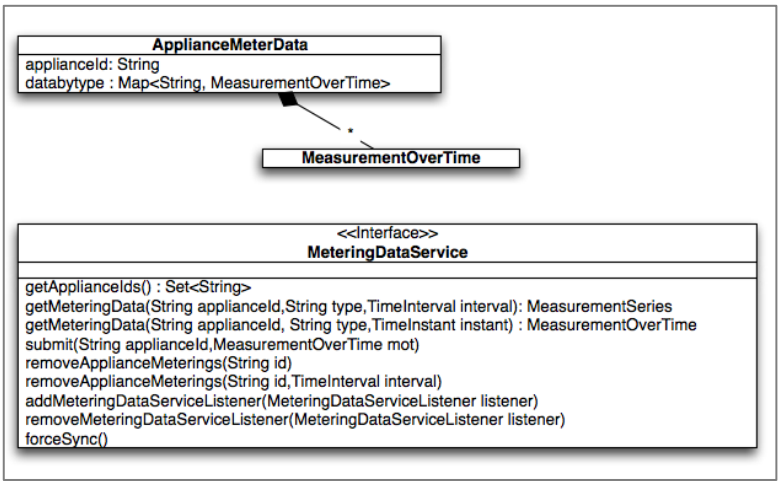


Figure 4-43 Example interface of a MeteringDataService

This design document does not elaborate further on much detail to realize this. However we wish to indicate that the provisions introduced here should have a solid, flexible but also lightweight realization. In general one can use concepts from the domains of data stream processing and/or complex event processing. In any case, we have transient data that is continuously updated, matching the DSMS (Data Stream Management System) concepts. However viewing information flow items as notifications of events, such as in the CEP (Complex Event Processing) domain also matches our purpose, since we have a decoupling between the framework and applications which prevents us from viewing our system as an entire data flow concept. One should realize a lightweight infrastructure, providing flexibility for data stream representation, including an event mechanism that Apps can subscribe to, and finally providing some persistence locally to the node and efficient sync to the Management Center of key facets.

The diagram below illustrates this correspondence, using the terminology from the EPTS (Event Processing Technical Society) glossary with event source, event consumers (Apps in our case), subscriptions to event channels providing event streams (or notifications) to its subscribers, based on an underlying data source receiving data from an event source via a receiver.

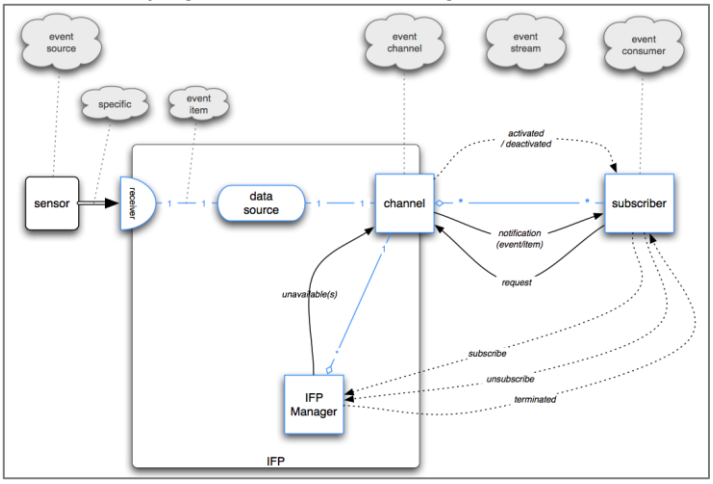


Figure 4-44 Relation with data stream and event processing

The IFP (Information Flow Processor) shown not only translates specific information towards general event items and stores them in a data source, but also provides channels to subscribe to the information.

4.2.2.13 *Composition of the Energy Application Layer*

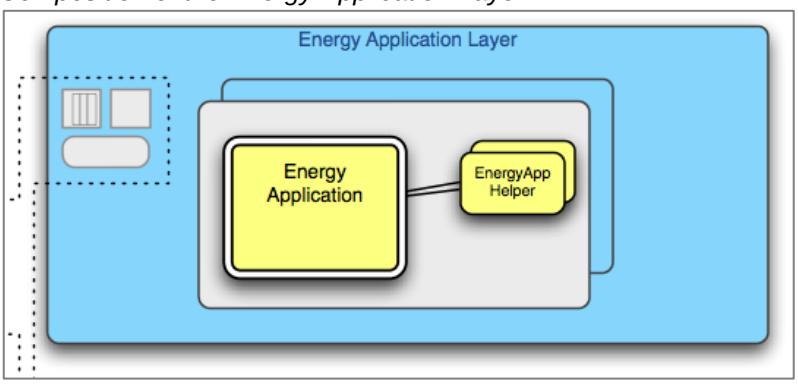


Figure 4-45 Composition of the Energy Application Layer

4.2.2.13.1 *ECS provisions*

The Energy Application Layer (EAL), as any decision layer, has ECS provisions; a MailBox, a Decision Logic Manager and the provision of a WorldView for its decision logic components. The decision logic component of the EAL is the EnergyApplication. There is at least one but there can be multiple ones.

4.2.2.13.2 *EnergyApplication*

An EnergyApplication is a PLUGIN, which can be replaced by an App. There can be multiple EnergyApps. An EnergyApp can have EnergyApp Helper Apps.

The relations between an EnergyApp, its Resources and its EnergyMarketParticipant were already introduced before. An EnergyApp works with none or one EnergyMarketParticipant (or multiple using the same protocol). An EnergyApp works for a set of Resources. If there are multiple EnergyApps, they are either in competition or they work with disjoint sets of Resources.

The EnergyApplicationLayer, in fact the EnergyApp WorldView provided by the ECS, has access to the static user energy targets, to the total energy consumption and generation information, etc.

Recall that the EnergyApp is supposed to make a decision in building a consistent set of EnergyProposals. In its decision cycle it should submit all EnergyProposals belonging to that set. When it previously submitted an EnergyProposal set {EP1,EP2,EP3} and in a new decision cycle decides to retract EP2 and submit EP4 in addition, it should submit EP2' which is EP2 with an indication of "retraction" and submit EP4; this to reflect the new set of proposals {EP1,EP3,EP4}.

The EnergyApp has a clear view on the energy consumption/generation situation of the account based on all past and current ControlSpaces and total consumption/generation information. It probably would be helpful for an EnergyApp to have a broad view on the situation in the market place, to take this into account for making EnergyProposals. This could be realized by making quick "trial" proposals to learn about the current market situation. Another option could be, e.g. by a EnergyApp Helper App, to build/offer an (estimate) market situation representation to the EnergyApp, so that it can use this in its reasoning. Both options are illustrated in the next figures.

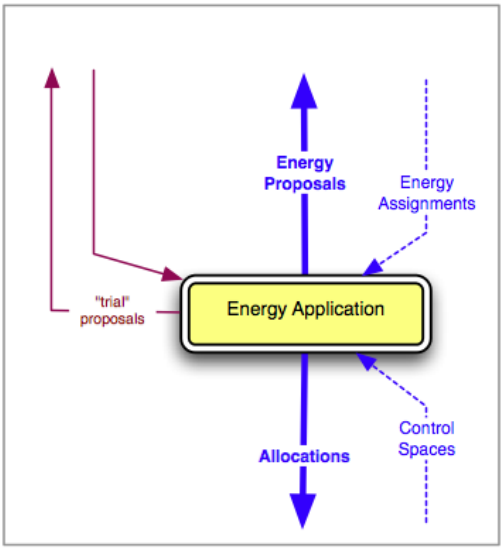


Figure 4-46 Quick view on market situation by "trials"

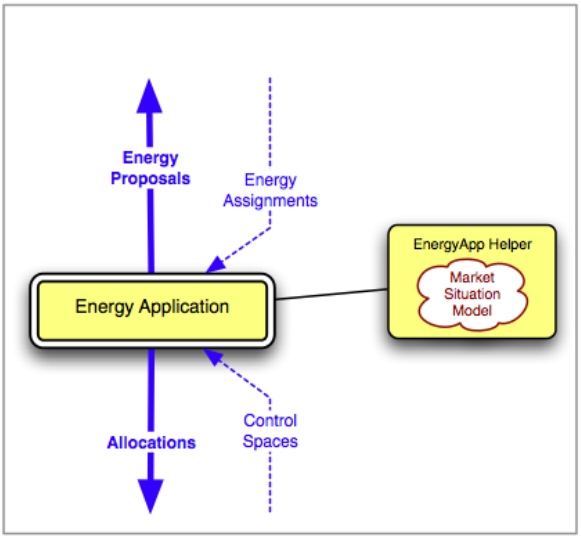


Figure 4-47 Market situation by Model in App

4.2.2.13.3 Energy Market Participants and Resources of an EnergyApp

An EnergyApp works with none or one EnergyMarketParticipant (or multiple using the same protocol). Multiple EnergyApps can realize the involvement of multiple EnergyMarketParticipants. This is shown in the next figures.

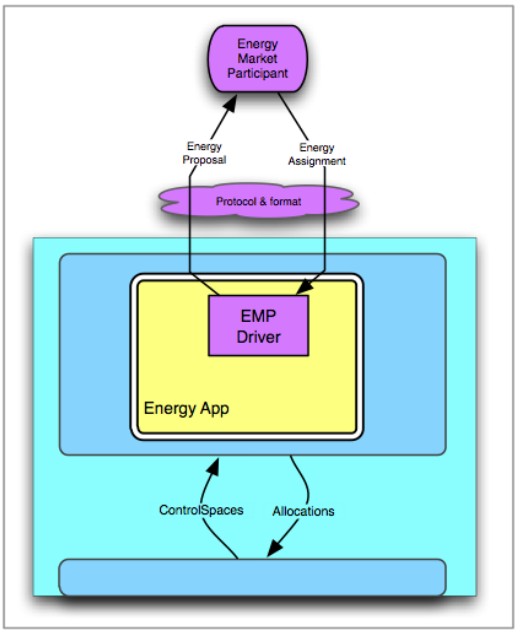


Figure 4-48 EnergyApp with 1 participant

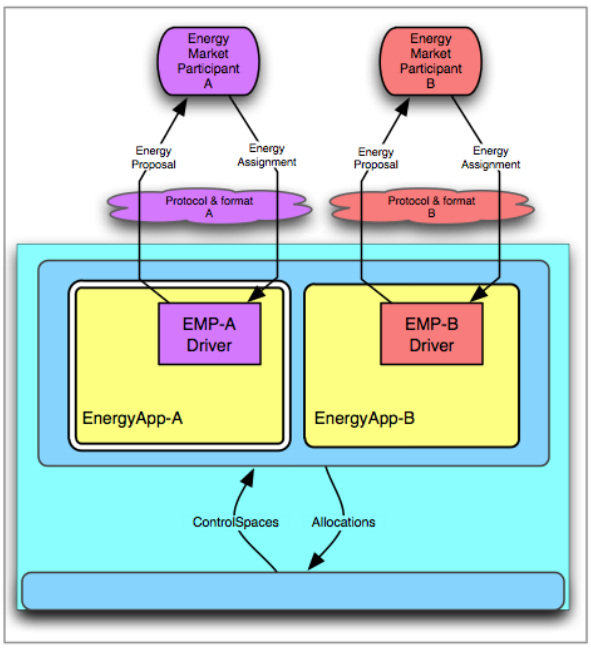


Figure 4-49 Two EnergyApps, two participants

An EnergyApp works for a set of Resources. If there are multiple EnergyApps, they are either in competition or they work with disjoint sets of Resources.

The relation between an EnergyApp and its Resources is specified as a ResourceSet of an EnergyApp. This relation is managed by the SMF.

The specification of a ResourceSet for an EnergyApp can make use of four ways to identify resources:

- ResourceFunctions
- ResourceCategories
- Guids
- Instance

This is illustrated in the figure below.

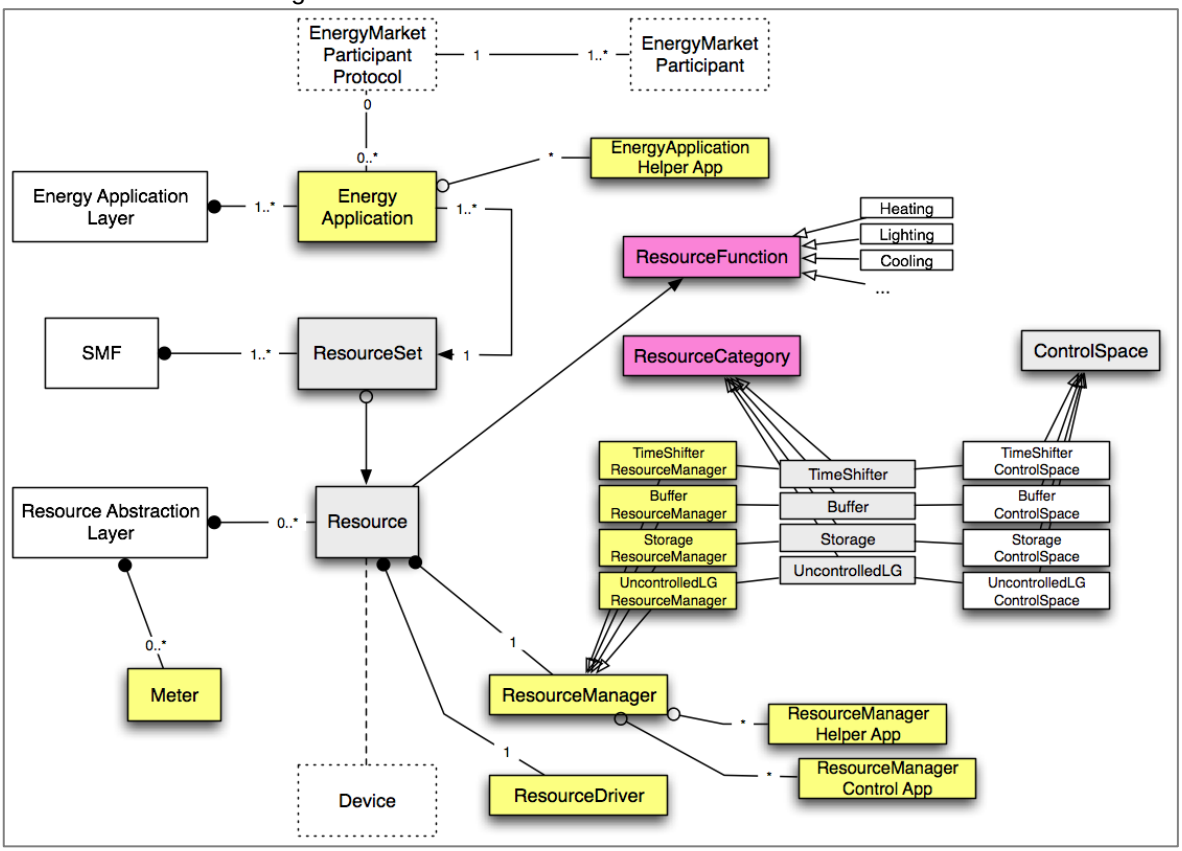


Figure 4-50 EnergyApp ResourceSet specification

A very specific way to identify a resource is by using the AppGuid of its ResourceManager. Remark that when two resources use the same ResourceManager App, one can even be more specific by specifying the instance of the ResourceManager. A more generic way is using the ResourceCategory, which reflects the energy flexibility type of the resource and corresponds with the top-level types of ResourceManagers and the types of ControlSpaces. A generic but functional identification consists of using a ResourceFunction. ResourceFunctions are defined by the user and assigned to resources to categorize them according to their function. In the figure above examples are given of ResourceFunctions such as heating, lighting, cooling, … One could allow the user to specify a hierarchy of functions and identify resources with one or more of these ResourceFunctions.

The ResourceSet of an EnergyApp can be defined by the User via the (Control) GUI. An EnergyApp and equivalently a ResourceManager App can have restrictions for which associations are allowed. One can image a business model to offer packages of devices linked with an

EnergyMarketParticipant for the supply of the electricity they consume. In such case one would offer AppGroups in the store with EnergyApp, ResourceManager and ResourceDriver Apps (and optional additional helper apps), where one has fixed or constrained ResourceSets for the EnergyApp.
Note that ResourceFunctions and ResourceCategories also come into play in relation with user energy target settings. As discussed before, so-called “static” settings are default profiles, which are typically associated with resources using ResourceFunctions. In this way one can specify a default energy target setting such as a week schedule for resources related with the “heating” ResourceFunction. So-called “dynamic” settings are deviations from the static settings for specific resources. One typically uses the more specific identifications (instance or Guid) to associate a setting with a specific resource.

4.2.2.14 Composition of the Interaction Framework

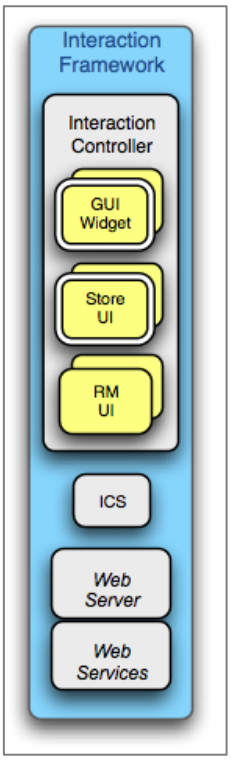


Figure 4-51 Composition of the InteractionFramework

The Interaction Framework provides all functionality for a User to control the Node using a UserDevice.

Interaction is about different aspects, including:

- Account management (adding/changing Users, UserDevices, Roles, etc.)
- Setting static user energy targets
- Setting dynamic user energy targets (alternatively on the device itself if possible)
- Visiting the FP AppStore and purchasing Apps
- Getting insight into the consumption/generation patterns

4.2.2.14.1 The ICS

The InteractionFramework acquires an authorization token from the Authorization Manager (of the Operations Manager of the SMF) so that it can independently work with the external control communication interfaces via the ICS scheme. This token-based authorization makes the InteractionFramework rather independent in its external communication, which has different synchronicity and visible response times (visible to the User with a UserDevice) than the other external communications.

4.2.2.14.2 WebServer, GUI and WebServices

It offers a Web-based GUI, which is served by its WebServer and consumed by a web browser on a UserDevice. It also offers WebServices to support three different aspects:

- **Information WebServices** for Users to ask Node information
- **Interface WebServices** to a client-based GUI on a UserDevice.
- **Sensor WebServices** to supply UserDevice sensor data to the Node.

Using Sensor WebServices, the UserDevice consuming the web-based GUI or offering the client-based GUI can integrate UserDevice sensor data. This can be used to integrate location and orientation sensor data to support the GUI for functionalities that use the position/orientation of the UserDevice in relation with the physical location of devices of the account.

Another use of Sensor WebServices is to feed sensor data of all UserDevices, not necessarily the controlling UserDevice consuming/offering the GUI, but also UserDevices such as smartphones of other Users of the account. One could supply location information of Users of the account via their smartphones to help with the energy management of the Node.

Sensor data is communicated via the Sensor WebServices is added to the general SMF Model, so that this information is not only accessible to control-related functionality but also for decision related functions, such as the decision logic of an Energy Application. In the example of location data, the Energy Application could take into account the location of all of the account users for example using geo-fencing to control energy devices.

Sensor-data can be GPS location data (useful to position the User in the account premise to support the GUI for walkthrough / virtual reality applications, to locate where all the account members are for using geo-fencing applications, to identify which building of a multi-location account the User consuming the GUI is located, etc.), compass and gyroscope data (to augment the location data with orientation in all directions), camera data (to support virtual reality or video overlay applications to show energy consumption information or to support QR-code scanning).

4.2.2.14.3 The Interaction Controller

The Interaction Controller provides default GUI control, but its functionality can be extended via three categories of Apps[45]:

- GUI Widgets
- Store UI Apps
- ResourceManager UI Apps

Apps can provide extensions to the GUI with **GUI Widgets**. For example one can offer rich insight into current and historic consumption/generation and provide help optimizing the user energy targets. There is default GUI Control in an FP Node, therefore there is a GUI Control plugin, which can be replaced by an App or receive additional GUI Widgets to extend its functionality.

A **Store UI** provides a richer experience to visit the FP AppStore than offered by the default functionality and GUI of the FP Interaction Controller. There is a default Store Control Plugin in an FP Node, therefore the Store Control is a Plugin, which can be replaced by an App.

[45] This design is of a functional nature only. We therefore label all regions of the system where additional functionality can be added as Apps or places where kinds of Apps can be deployed; and introduced the notion of AppGroups to have logical sets of functionalities. In the technical design, these notions are much more refined. Reference Implementation 12.11 for example has Apps which can have multiple modules, an App is provisioned as a set of jars (OSGi bundles), where some can explose ConfigurableServices and factories of these services to deploy Resouce Managers, Resource Drivers, etc. In addition, these can register Widgets to offer GUI interaction. Since the scope of this document is functional design only, this more refined terminology is not introduced here.

A **ResourceManager UI** is an App that works with a ResourceManager to offer specific GUI functionality related with the Resource. It can for example offer a specific GUI page to control and view the specific controls and settings of a device.

Since devices are abstracted by ResourceManagers of a number of types according to the ontology (buffers, time shifters, etc.), the default GUI is limited in offering views and controls specific for the device. Such a RM UI app can offer more specific control functions.

Note that an AppGroup typically combines the ResourceManager App, the ResourceDriver App and the ResourceManager UI App.

The design does not provide for additional Helper apps for UI apps.

- If a ResourceManager UI App would like to use a Helper App to provide it with useful information, it can use the ResourceManager Helper App, which are already allowed in the RAL to be associated with the ResourceManager. Since the ResourceManager UI App already communicates with its ResourceManager App, it can receive the information via that way.
- A Store UI App is related with the Store and less with the Node itself. If it would like to use additional help, it should receive this via the store backend[46] that supports it.
- It could be helpful for a GUI Widget App to receive additional help. If this help is related with GUI technical components (e.g. a 3D rendering component), then this functionality is provided as Interaction functions or components instead of Apps. If the help is related with information provision, the GUI Widget App should either subscribe to an information feed or by itself receive information from a server-side part; we do not foresee the provision of information from additional helper apps.

[46] This is called the “Store Provider” as we will see in the decomposition of the store further in this design document.

4.2.3 *FP ManagementCenter*

The FP ManagementCenter is a system component that the MCO operates to manage the Nodes in its FP Net. Its behavior is already determined by the external communication interfaces it offers. We describe a high-level functional decomposition to identify its provisions and representations needed to fulfill its responsibilities.

4.2.3.1 *External Communication Interfaces of an FP ManagementCenter*

The external communication interfaces that an FP ManagementCenter (MC) must realize were already introduced. The next figure shows an overview.

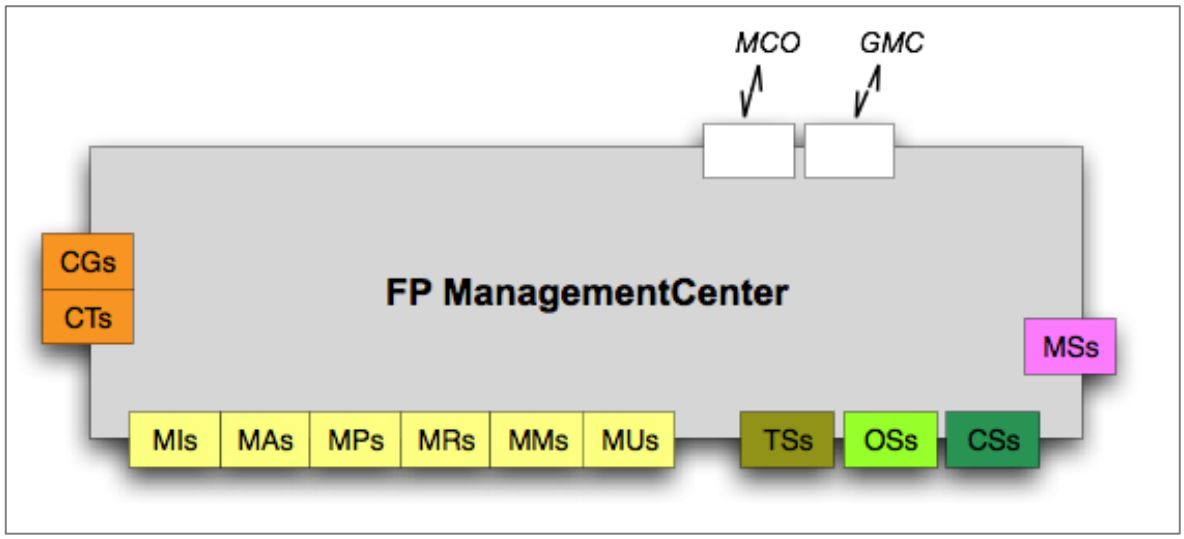


Figure 4-52 MC with external communication interfaces

The communication interfaces to interact with the MCO and GMC were not formally introduced yet.

For a Management Center Operator (MCO) to run its MC, it needs a control interface to interact with the MC internal systems. This interface is not described in this design specification.

As already discussed, there can be a hierarchy of management centers, where an FP GroupManagementCenter (GMC) manages either different MC's or different GMC's. When this group management involves system interactions, the MC needs an external communication interface to interface with its GMC. This interface is not described in this design specification.

4.2.3.2 *Components of an FP ManagementCenter*

The next figure summarizes the components of an FP ManagementCenter. These are functional components that do not necessarily all have to be translated into technical systems and components. However, they illustrate the provisions and representations needed by a MC to realize its responsibilities.

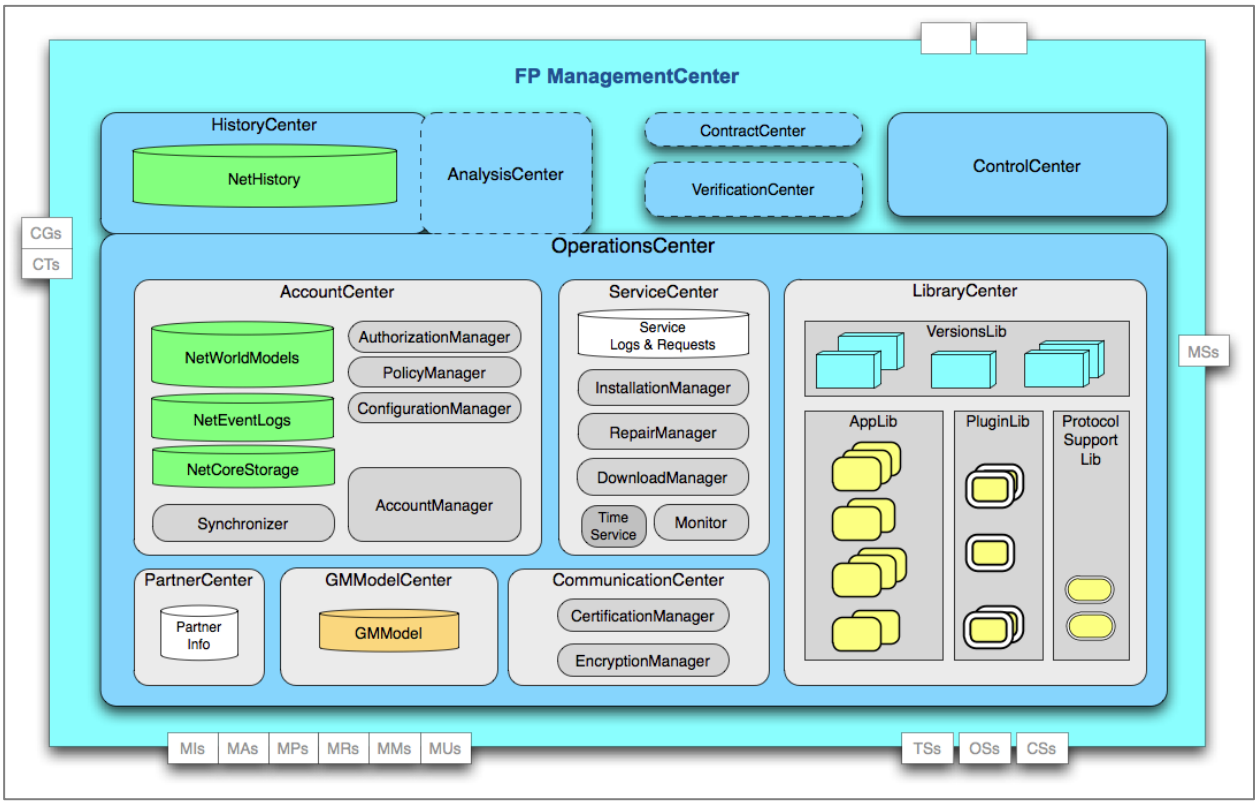


Figure 4-53 Functional decomposition of an FP ManagementCenter

A FP ManagementCenter (MC) has one major operational component, the OperationsCenter.

As previously mentioned, Nodes are quite independent and can continue their operation when the MC is unavailable.

The MC is considered available when its OperationsCenter is available, however its HistoryCenter and AnalysisCenter can also be involved in the operations, so their availability also determines partly (see further) the full availability of the MC.

The other components of a MC are support components that are not critical for the operational duties of a MC for its FP Net accounts and their partners.

4.2.3.2.1 *Operations Center*

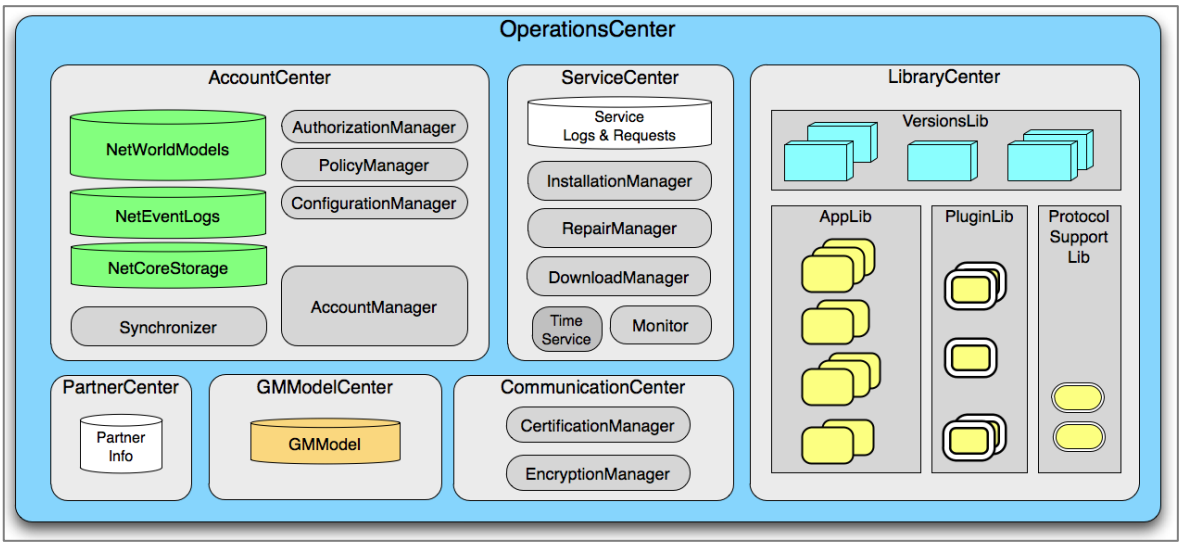


Figure 4-54 Components of the Operations Center

The OperationsCenter performs the management of the accounts and their nodes with its **AccountCenter**, it keeps management information about other parties in the FP Net at its **PartnerCenter**, it performs communication services via its **CommunicationCenter**.

The **ServiceCenter** collects all services related with installations, monitoring, repair/restore/updates. It also provides the download services for Apps purchased in the FP AppStore.

Its **LibraryCenter** is the repository of all software versions, installed Apps, available Plugins and Protocol Connectors.

Nodes have a SMF Model with three parts (WorldModel, EventLog, CoreStorage) that are subject of syncing with the MC, it has a fourth part is a local management model not subject to syncing with the MC (the SMModel with local security and management information).
Similarity, the MC has the state of all account nodes collected in three models (NetWorldModels, NetEventLogs, NetCoreStorage) which are filled by syncing operations from the nodes, its has also a fourth part which is the global management information model (GMModel).

4.2.3.2.1.1 Account Center

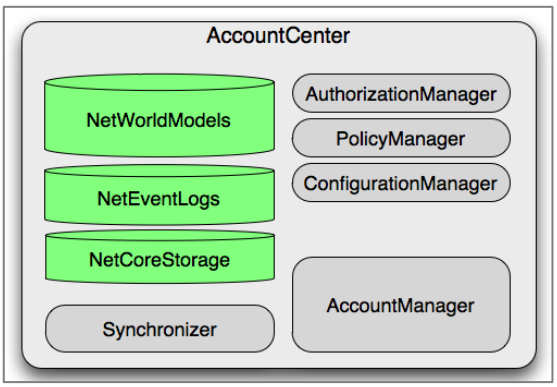


Figure 4-55 AccountCenter of OperationsCenter of MC

The **AccountManager** coordinates the functions of the AccountCenter. The AccountManager is responsible for handling all requests via the M* external communication interfaces.

The “state” of all account nodes is collected via the **Synchronizer** into the **NetWorldModels**, **NetEventLogs** and **NetCoreStorage**. These have the same structure as the equivalent parts of the SMF Model of each Node, but combined now for all accounts.

The Synchronizer works in coordination with the SMF of the account Nodes to handle the incremental synchronization requests. The sync management information is part of the GMModel of the GMModelCenter.

The **ConfigurationManager** manages configurations of Nodes; this to determine which combinations of software, Plugins and Apps are valid. The node configuration information is part of the configuration model part of the NetWorldModel, similarity as the node also keeps its configuration information in the configuration model part of the WorldModel of the SMF Model.

The **Authorization Manager** manages authorization requests from Nodes. The authorizations and role definitions are part of the GMModel. Note that granted authorizations can be revoked.

> As an example, when a node asks to active a purchased and downloaded App, it asks authorization. The authorization manager will check the verification of the App, as provided by the Verification Center. In addition it will check with the Configuration Manager if the App is a safe combination for the current configuration. It could be an invalid combination, or a combination that the MC learned is giving problems from monitoring other nodes, or it could require additional components installed first such as ProtocolConnectors in the case the App requires protocol support that is not installed yet.

The **PolicyManager** manages policies, which are part of the GMModel. The MC can change policies and communicate this to the account nodes for compliance.

4.2.3.2.1.2 Partner Center

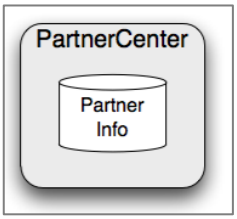


Figure 4-56 The PartnerCenter of OperationsCenter of MC

The PartnerCenter keeps information about all other partners in an FP Net, besides the accounts. It has a **PartnerInfo** model to store this information. The PartnerInfo model contains the credentials of all Partners that wish to participate in the FP Net. These credentials are needed for handing out certificates to communicate with account nodes. Note that the certificates are kept at the GMModel. The PartnerInfo model is consulted when nodes ask discovery requests to find partners. Note that therefore the credentials of a partner must include what they can offer, in detail its must include DiscoverySpecifications as introduced in chapter 3.

4.2.3.2.1.3 GMModel Center

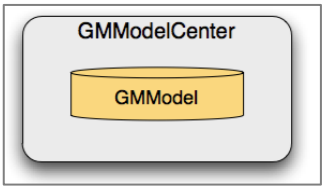


Figure 4-57 The GMModelCenter of OperationsCenter of MC

The GMModelCenter keeps the global management and security information for the MC as a **GMModel** to manage the FP Net. The information structure is highly similar to the SMModel of a SMF of an FP Node, which is the local management model of the node.

The information structure is shown in the diagram below.

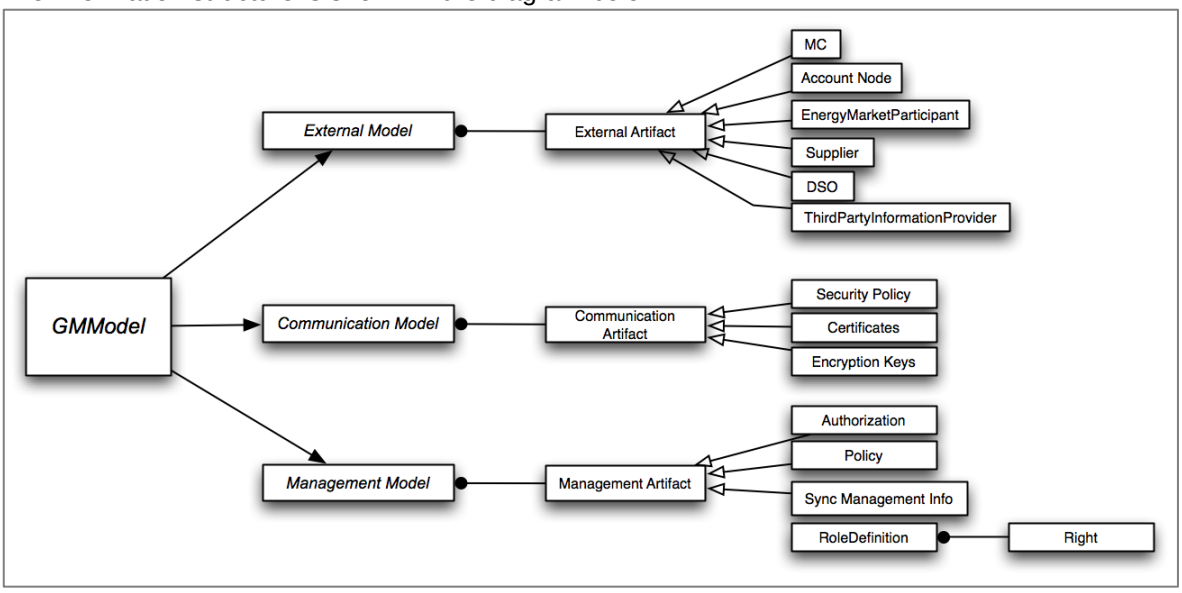


Figure 4-58 Information structure of the GMModel of MC

It contains an external model with abstract representations of all parties, used by the CommunicationCenter and the MC external communication interfaces to address these parties.

It has a communication model to keep all communication artifacts such as security policies, certificates (valid and revoked ones), encryption keys etc.
It has a management model with all global management artifacts needed by the OperationsCenter, such as authorizations (valid, refused and revoked ones), policy definitions (current active policies and previous inactive ones), management information to keep track of syncing operations with the nodes and definitions of roles with their rights (current definitions and previous ones).

4.2.3.2.1.4 Communication Center

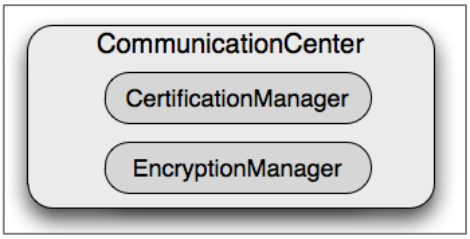


Figure 4-59 The CommunicationCenter of OperationsCenter of MC

The CommunicationCenter not only works with the external communication interfaces of the MC to communicate, but also contains the certification authority system of the FP Net, called the **CertificationManager**. It also has an **EncryptionManager** in charge of management of encryption keys and services.

The CertificationManager handles requests for certificates or renewals by checking the party credentials for partners with the PartnerCenter and for accounts with the AccountCenter.

4.2.3.2.1.5 Service Center

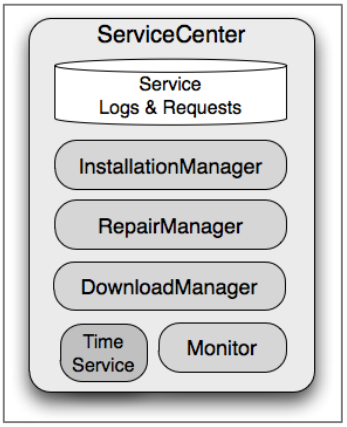


Figure 4-60 The ServiceCenter of OperationsCenter of MC

Its ServiceCenter provides the service-related functionalities of the MC.

These services not only include installations (**InstallationManager**) and repair/restore/update (**RepairManager**) functions. There is also a **DownloadManager** that handles the download of Apps purchased by an account from the FP AppStore.

The ServiceCenter has all the component information and sources available from the LibraryCenter, so that it is independent from the availability of the store to perform install/repair/restore actions or to provide the download of purchased apps.

The **Monitor** component is responsible for keeping track of the health of the FP Nodes (on FP HomeBoxes). On the one hand it can use the system-level monitoring functions via the CG external communication interface (TR-069). On the other hand, it can monitor the EventLog data coming in via sync operations and available in the NetEventLogs, since these also reveal a lot about the node health.

The ServiceCenter also has a **TimeService**, which provides the synchronized time-clock function that parties in the FP Net can synchronize on via the TS external communication interface.

4.2.3.2.1.6 Library Center

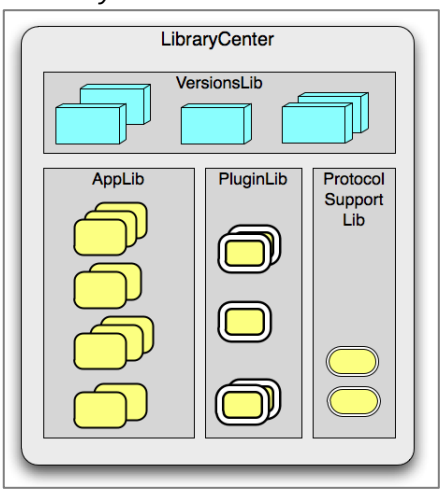


Figure 4-61 The LibraryCenter of OperationsCenter of MC

The OperationsCenter has a LibraryCenter to keep all the info and sources of all software versions, apps, plugins and protocol support components.

These are used by the ServiceCenter for installations/repair/restore/update actions, as well as by its DownloadManager to provide the download for purchased apps. Recall that upon purchase, the FP AppStore first makes sure that the MC has already or acquires the App and consequently delegates the download process to the MC before committing the purchase to the node.

The LibraryCenter has all the versions of everything, not only the last versions. Even if for example the store only offers a new version of an App, the LibraryCenter has to keep older versions if they are still used by some account in its FP Net.

For Apps and Plugins, the ServiceCenter can provide updates/installations via the OS external communication interface. For installing/updating software versions or installing some of the protocol support, the CG interfaces are needed since the updates (could) require node restarts.

This can be the case for protocol support updates. The Protocol Support Lib contains ProtocolConnectors, for which an installation is not required to be possible on a live system without requiring a restart.

4.2.3.2.2 History Center

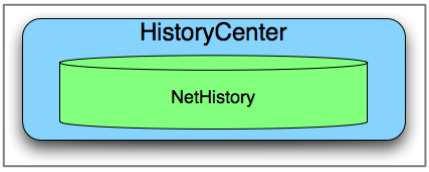


Figure 4-62 The HistoryCenter of MC

The state of all account nodes is represented by the three models in the OperationsCenter, which fill up by sync operations from the nodes. To offload older information that is no longer required for the main operational functions of the MC, the information is stored as the **HistoryCenter**.

However the HistoryCenter is operationally important for two reasons

- To help resolve information collection requests that span the older history
- To support the AnalysisCenter if the MC would decide to provide that service (see further).

4.2.3.2.2.1 Information Collection Requests could propagate into the HistoryCenter

Recall from the introduction of the external communication interfaces and the design of an FP Node that a SMF (its Data Access Manager or its Operations Manager) can resolve requests for information by looking into the local data models (the SMF Model), when the window of the request would exceed the local scope it can forward and InformationCollectionRequest to the MC (secondary nodes can forward requests in a similar way to its primary node). Via the MI external communication interface the AccountManager (of the OperationsCenter of the MC) handles such request. It can consult the broader[47] window of information from its NetWorldModels and NetCoreStorage (in theory also from its NetEventLogs although collection of information probably never requires extraction from event data). When even that window of information is not sufficient, it can consult the HistoryCenter to collect information from the entire history window.

For nodes, either to visualize feedback on their GUI or to drive their models, access to historic information can be important. For example, when one wishes to show in the GUI a curve of consumption over time that needs information beyond the window available in the node, one resolves the part exceeding the local window by an InformationCollectionRequest to the MC. The OperationsCenter AccountManager can resolve the part covered by the models of the OperationsCenter and request information collection to the HistoryCenter for the part not covered by its own models.

4.2.3.2.2.2 Example of full propagation

As an example of a cascade of information collection requests, consider the following example.

When the GUI of a node wants to display an energy consumption curve of a specific resource over a broad time window, its RM UI App that provides a specific GUI page for the resource renders the curve, based on data it asks from its associated ResourceManager App. The ResourceManager App consults its WorldView for the data collection, which internally is provided by the Data Access Manager of the SMF. It resolves part of the time window by consulting the SMF Model and resolves the remaining part by forwarding an InformationCollectionRequest to the MC. The AccountManager of the MC resolves the request by consulting its models and resolves the part of

[47] Broader is not the correct term, since the coverage of data models actually has a Venn-diagram like structure, as shown further in an example diagram in the next paragraph.

the time window not present any more in these models by forwarding a request to the HistoryCenter. In a situation where the resource is a remote resource controlled by a secondary node, the ResourceManager of the secondary node is involved so that we could even have four parties involved in resolving a single request. This propagation example is illustrated in the diagram below. It shows the parties involved, the data models (green) consulted and the sequence of requests by boxed numbers. The lower part of the diagram shows the coverage of each data model and the window from which information is collected in each consultation step.

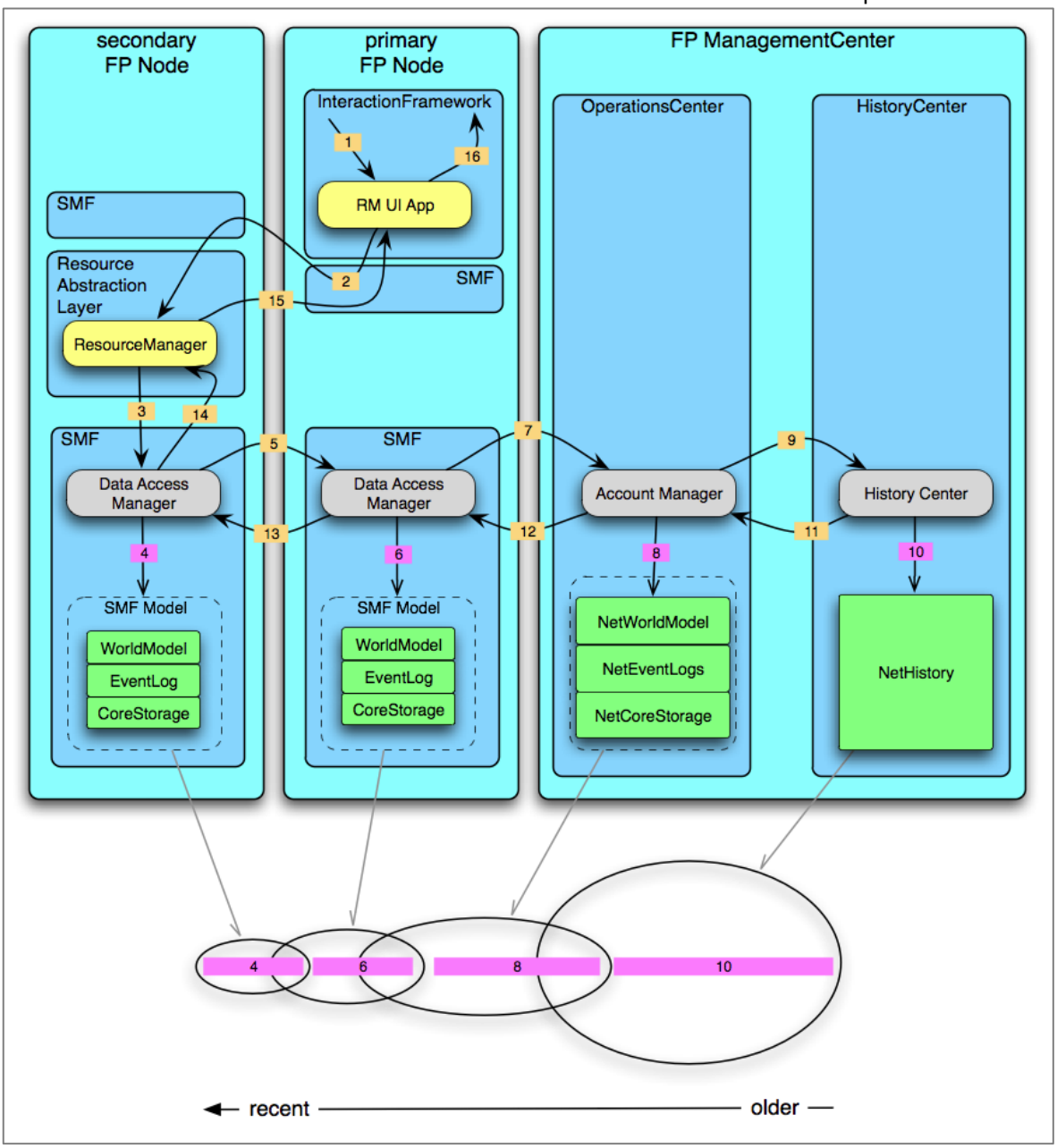


Figure 4-63 Example of propagation of information collection requests

Note that the data models have indeed the Venn-diagram structure of coverage as shown in the diagram, since a local model can have fresh information not synced to the higher model, common information already synced with the higher model and information not locally present any more but synced to a higher model or further.

4.2.3.2.3 *Analysis Center*

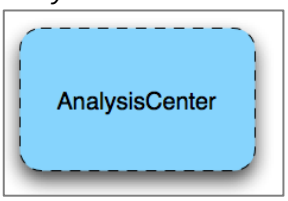


Figure 4-64 The AnalysisCenter of MC

The information collection propagation method discussed before has two restrictions. First is only considers information about the account, not across accounts. Secondly the processing of information happens in local components of the node, thereby limiting the learning/complexity of the processing models due to performance restrictions.

4.2.3.2.3.1 *Open Analytic services*

To open richer possibilities for analysis/analytics components, the MCO could provide access to historic information, even across accounts, to other parties. Given that the proper privacy/security/ terms-of-use contracts are in place, other parties, such as server-side components of apps, could access the historic information to drive their learning/analytics/forecasting models.

These analytics services are provided by the AnalysisCenter, which either provides access for or even could host systems that want to exploit the information.

An AnalysisCenter can access the oldest information from the HistoryCenter and the current information from the AccountCenter. It cannot access the most actual information, which is only local for the Nodes, however Nodes sync their information regularly, so there is only a very small slice unavailable for analysis.

This AnalysisCenter component is optional and not a necessary part of an operational MC, therefore a dotted borderline is used. It becomes an operational part of a MC if the MCO wishes to provide these services. Note that the external interface for other parties to access the AnalysisCenter is not depicted in the diagrams used to sketch the FP ManagementCenter.

4.2.3.2.3.2 *Analysis by simulation*

The AnalysisCenter can also provide simulations based on the detailed and structured information the ECS of nodes provided. The availability of the EventLog information and the fact that the ECS was designed to have a strict coordinated operation, ensuring a correct and consistent environment for decision cycles of the decision components of a node, both ensure that simulations of past node operations can easily be performed. The simulations can be visualized and reviewed even without having to execute the decision cycles of the decision components, since all input and output decisions of all decision components were captured. Simulations are not sensible to concurrency side effects, since the environment for each decision cycle was made consistent and captured by the ECS. Thereby analyzing a simulation at higher speeds is rather straightforward.

Simulations like this can be interesting for the MCO to gain insight. They could even be interesting for accounts to gain insight about past events and their benefits. Simulations like this could also be used to measure what alternative decision components would have improved considering the past or to match different account to classify them into profiles. Open Analytic services as discussed in the previous paragraph could enable App developers to provide information like the above with a server-side analytics component.

4.2.3.2.3.3 *Analysis for App testing*

The MC could use its AnalysisCenter to provide offline testing of new Apps for App developers. Making use of the rich information from the History Center, testing Apps using simulations with this data would be very useful for the developer community. In a similar way an FP AppStore could use the AnalysisCenter to perform tests as part of their App approval process, the MCO could use the AnalysisCenter to perform tests as part of their App security verification process. Note that specific attention should be paid to assure that there is no interference between operational duties of an Analysis center and these testing facilities.

4.2.3.2.4 *Control Center*

Figure 4-65 The ControlCenter of MC

The ControlCenter is the component of the MC that is involved to provide an interface to the MC for the MCO. It allows the MCO to connect a system/GUI to the MD.

4.2.3.2.5 *Verification Center*

Figure 4-66 The VerificationCenter of MC

All software components, plugin and apps need to be verified from a security and system stability perspective. The FP AppStore is responsible for the approval process of an App, but even an approved App needs a security and system stability verification from the MC.

The business process at the MC to perform these verifications is the VerificationCenter.
This is not necessarily a software entity; therefore a dotted borderline is used.

4.2.3.2.6 *Contract Center*

Figure 4-67 The ContractCenter of MC

The ContractCenter is a non-operational component; it is even not necessarily a system component (therefore the use of a dotted borderline) but could be a separate business process, which is responsible for managing all contracts concerning accounts and partners of an FP Net.

Besides the functional components, representations and provisions described, an FP ManagementCenter needs contractual agreements to ensure all of its responsibilities.

Besides contractual agreements regarding use of FP Application Infrastructure platform software, contracts with its Accounts and contracts with the FP AppStore and its operator, the following contractual provisions are if high importance:

- Since the MC stores sensitive information about energy usage and management of its accounts, the privacy aspects of this information should be assured.

    - It is necessary for the MC to store the account energy management information to realize its management responsibilities with respect to installation/repair/restore actions.
    - The FP Application Infrastructure platform has provisions to protect this data at the Node level and communication level with the MC. It has provisions for authentication (certificates-based) and security (encryption of communicated information).

- To protect the privacy of the data at the MC level, the MCO should underwrite an agreement to protect the collected information with necessary technical measures and not forward the information to third parties outside the agreed terms of use.

- The MCO needs to make sure that it has consent of its accounts to store the information and its terms of use.

- The MCO should assure that the FP AppStore has
    - Proper approval procedures in place to approve Apps
    - Proper protection procedures and systems in place to protect the account information it obtains from the MC to realize its invoicing
    - Proper protection procedures and systems in place to protect the configuration information it (could) obtain from Nodes. Note that Nodes can provide (limited) configuration details to help the store in optimizing its presentation and filtering of Apps for the account users (e.g. by using the already supported device protocols in the view/filtering of available Apps).

- The MCO should have proper contractual agreements in place for partners to participate in the FP Net communication, so that the privacy and terms of use of the exchanged information is bounded. This with respect to EnergyMarketParticipants receiving energy proposals, App developers in case their App collects information or interacts with the deployed App via a server-side component, ThirdPartyInformationProviders in general in case their information feed also includes communication back from the Nodes.

- In case the MC allows access to its current or historic information for analysis/analytics purposes, it needs to assure that proper data anonymization procedures are applied and that this use is part of the consented terms of use. Analysis/analytics applications can involve offline analysis (where information is transferred to a third party) or online analysis (where a third party uses a system to link to the MC current and/or historic data, for example to drive forecasting/learning models). A MC could support online analysis by hosting analytics applications or providing them with access.

Since the MCO with its MC is the managing authority for the FP Net, it is the primary responsible party to secure the contractual agreements and consents.

4.2.4 *FP AppStore*

The AS is a system component that offers an application store for the FP Net.

The FP AppStore (AS) is a system component that the ASO operates to offer an application store for an FP Net.

Its behavior is already determined by the external communication interfaces it offers.

We describe a high-level functional decomposition to identify its provisions and representations needed to fulfill its responsibilities.

4.2.4.1 *External Communication Interfaces of an FP AppStore*

The external communication interfaces that an FP AppStore must realize are shown in the next figure.

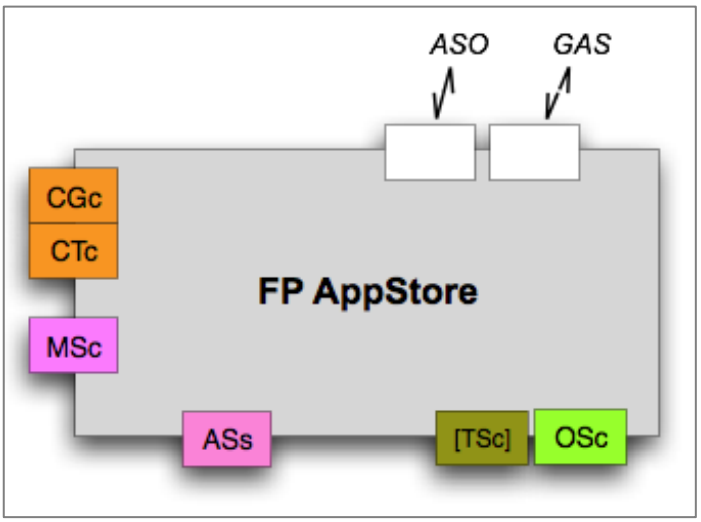


Figure 4-68 FP AppStore with external communication interfaces

The interfaces were already described before, except for the top ones in the figure that interact with the ASO and GAS. For an FP AppStore Operator (ASO) to run its AS, it needs a control interface to interact with the AS internal systems. This interface is not described in this design specification. There can be a hierarchy of application stores, where an FP GroupAppStore (GAS) manages either different AS's or different GAS's. When this group management involves system interactions, the AS needs an external communication interface to interface with its GAS. This interface is not described in this design specification.

4.2.4.2 *Components of an FP AppStore*

The next figure summarizes the components of an FP AppStore.

These are functional components that do not necessarily have to be translated into technical systems and components. However, they illustrate the provisions and representations needed by an AS to realize its responsibilities.

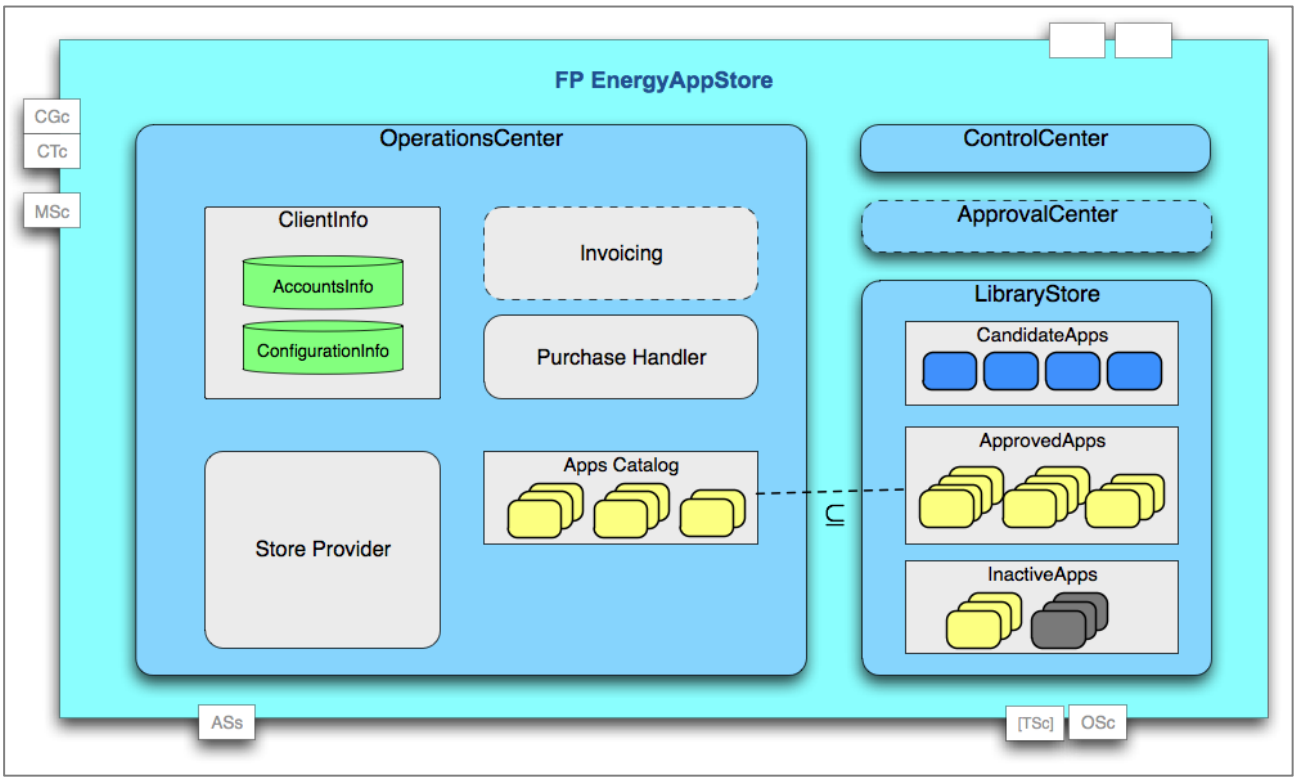


Figure 4-69 Functional decomposition of an FP AppStore

The FP AppStore has no critical availability requirements for an FP Net, the FP Nodes (on FP HomeBoxes) and FP ManagementCenter are highly independent from the availability of the store.

The store is considered available when its OperationsCenter is available.

4.2.4.2.1 *Operations Center*

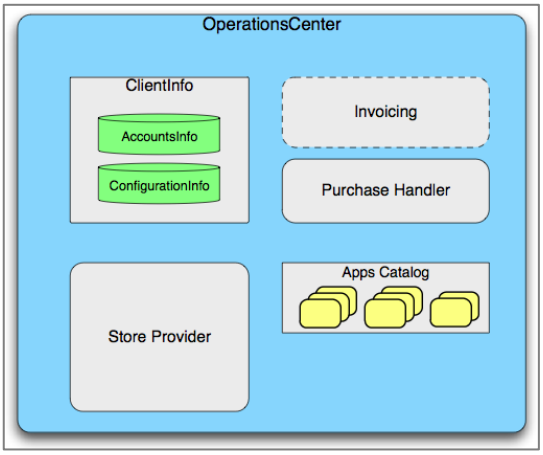


Figure 4-70 OperationsCenter of store

4.2.4.2.1.1 *ClientInfo*

The Operations Center of the store can collect limited information from its Clients in its **ClientInfo** component. This has **AccountsInfo** by asking account details to the FP ManagementCenter. This avoids a separate login procedure for accounts of the FP Net to access the store and it provides the store with the necessary information to invoice the accounts for purchases. It can also keep **ConfigurationInfo**. This is limited configuration information provided by the account Node to help the store to adapt the view, ranking and filtering of available apps, for example highlighting the Apps that rely on already supported protocols of the Node.
It stores the information in the ClientInfo component and needs to take the necessary security provisions as agreed on in the contract with the MCO.
Recall that the store and the accounts communicate via certificates, provided by the FP ManagementCenter, so only certified accounts can access the store and for those the store can ask client information to the MC.

4.2.4.2.1.2 *Store Provider*

The OperationsCenter has a backend infrastructure to provide the store experience, which is called the **Store Provider**. Either the default control GUI of the node, or a richer store App provided by the ASO and also made available (probably for free) in its store, provides the user experience of visiting the store and making a purchase.[48]

4.2.4.2.1.3 *AppsCatalog*

The Store Provider offers the backend for the store experience making Apps available for purchase. The Apps offered are contained in the **AppsCatalog**. It is a subset of all approved apps, as they are collected in the Library Store. The AppsCatalog is part of the OperationsCenter of the store since operationally important. The LibraryStore is not part of the OperationsCenter.
Every App and AppGroup of the AppsCatalog has a unique identifier (AppGuid). That identifier stays unique over time and is used in the FP Application Infrastructure platform to identify Apps and AppGroups by the nodes, the MC and the store.

[48] The store provider can be realized for example by a REST XML/JSON interface, which is consulted by a Store UI App in the InteractionFramework of an FP Node; where the app offers HTML for the user. One could consider making the backend interface of a PMEnergyAppStore public to allow application developers to interface with the store. However, this needs further consideration. One can also keep the interface private, and consider the communication between the Store UI App and the Store Provider a private matter in protocol and formats.

4.2.4.2.1.4 *Purchase handler*

The **Purchase Handler** is a component of the OperationsCenter of the store that handles the entire purchase.
As already discussed when introducing the external communication interface AS, this includes a communication with the MC to make sure that the MC has the purchased App sources before confirming the purchase to the account node. This is necessary to realize the independency of the MC. When the purchase confirmation is send to the account node, the Purchase Handler already delegated the download process to the MC, so that the availability of the store is no longer required for the download and installation process.

4.2.4.2.1.5 *Invoicing*

The **Invoicing** component of the store is responsible for invoicing the account for the purchase. This is an optional component, since the ASO could use its own business systems to provide this function. Therefore the Invoicing component is shown with a dotted borderline.

4.2.4.2.2 *Control Center*

Figure 4-71 Control Center of the store

The **ControlCenter** is a component that either provides a GUI or provides access to ASO systems to control the store.

4.2.4.2.3 *Approval Center*

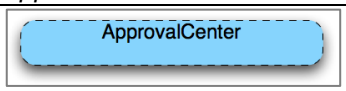


Figure 4-72 ApprovalCenter of the store

The store is responsible for approving apps before offering them for purchase. The approval process is situated in the **Approval Center**. This is not necessarily a system component but rather a business process. Therefore is it shown in diagrams with a dotted borderline.

4.2.4.2.4 *Library Store*

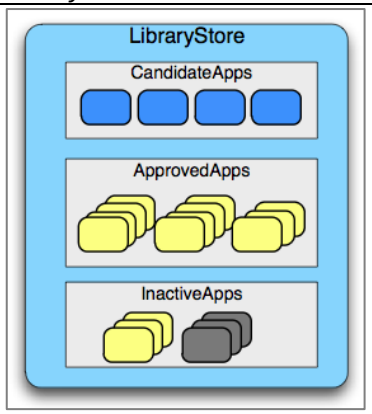


Figure 4-73 LibraryStore of the store

The **LibraryStore** is the component of the store where all Apps are collected. This is not an operational component, since the Apps offered in the store are also kept in the Apps Catalog of its OperationsCenter.

The LibraryStore keeps the candidate apps in its **CandidateApps** component, are they are received from App developers. When the ApprovalCenter approves an App, it moves to the **ApprovedApps** component. Note that different versions of an approved App are kept, as long as they can be subject of offering. Approved Apps that are no longer part of the catalog of the store are kept in **InactiveApps**. These can be Apps, which are still in use by some of the FP Net accounts, as well as Apps that are no longer used by anyone.
Note that AppsCatalog ⊆ ApprovedApps.

The LibraryStore has the responsibility to assign a unique identifier for the approved Apps and AppGroups. These so-called AppGuids must be unique over time and are used not only by the AppsCatalog but also by the node and MC.

4.3 Apps from an FP AppStore, categories and functionalities

In this section we elaborate on different categories of Apps that can be offered in an FP AppStore. We also briefly describe other possible offerings of such a store. We review different functionalities one can realize with Apps and how these interact with the node and other FP Net parties. We conclude with typical interaction patterns of Apps.

4.3.1 Categories of FP Node Apps

In its store catalog an FP AppStore can offer different types of FP Application Infrastructure Apps.

The table below lists the different categories of Apps a store can offer.
Besides apps of these categories, it also offers AppGroups that are combinations of different Apps.

FP Node Apps		
Group	*App Category*	*Description*
Interaction Controller Apps	GUI Widget	App to provide additional GUI functionality, as part of the overall GUI provided by the InteractionFramework of the Node. Deployed in the Interaction Controller of the InteractionFramework of the Node.
	Store UI	App to provide a rich GUI to visit the store, replacing the default store GUI as provided by the InteractionFramework of the Node. Deployed in the Interaction Controller of the InteractionFramework of the Node.
	RM UI	App to provide a GUI page with specific information and control for specific energy devices. Works in cooperation with a ResourceManager app for that specific device. Deployed in the Interaction Controller of the InteractionFramework of the Node.Typically part of an AppGroup together with a ResourceManager app and a ResourceDriver app.
Energy management Apps	Energy App	Energy App to replace the default Energy Application plugin of the Node, or to provide additional Energy Application(s). Deployed in the Energy Application Layer (EAL) of the Node.
	EnergyApp Helper	Helper app to help the Energy App. Deployed in the EAL of the Node next to an Energy App.
Resourcing Apps	ResourceManager	ResourceManager app to represent a resource. Deployed in the Resource Abstraction Layer (RAL) of the Node (primary or secondary). Works with a ResourceDriver app to monitor/control a device. Typically part of an AppGroup together with a ResourceDriver App.
	ResourceDriver	ResourceDriver app that contains the device/protocol specific logic to interact with a device. Deployed in the RAL of the Node (primary or secondary). Works with a ResourceManager app. Typically part of an AppGroup together with a ResourceManager App.
	ResourceHelper	Helper app to help the ResourceManager. Deployed alongside a ResourceManager in the RAL of a node (primary or secondary). Typically part of an AppGroup together with a ResourceManager app and a ResourceDriver app.
	Meter	App that provides information from a smart-meter via its protocol to the FP Runtime, so that other components and apps can use this via a MeteringDataService.

Apps can have a server-side. The server-side can supply them with information via the communication interface for ThirdPartyInformationProviders. This could imply that Apps could be purchased together with a license for using the information for a given period or volume.

Energy Apps are allowed to directly communicate with their server-side part.[49]
Note that ResourceManager and ResourceDriver functionality could also be embedded in appliances, described before as “embedded resources”.

4.3.2 *Other applications or tools that an FP AppStore can offer*

Besides Apps and AppGroups for the FP Node software of an FP HomeBox, a store can also offer applications or tools that are not to be installed as part of the FP Node software on FP HomeBoxes, but have some relation with the FP Application Infrastructure platform.

The table below lists different types of offerings for non-FP Node applications and tools.

Other Applications		
Group	*Application types*	*Description*
User Device Apps	iOS Apps	iOS app that provides a client-based GUI by interacting with the WebServices of the FP Node. To be deployed on an iOS device (iPad, iPhone). Typically offered in the store by its link to the iOS App Store.
	Android Apps	Same but for Android. To be deployed on an Android device. Typically offered in the store by its link to an Android App Store.
PC applications	PC/Mac/… applications	PC application using information from the Node via its WebServices to provide a GUI. To be installed on a PC/Mac/…
Partner Tools	Participation toolbox	Software toolbox to be used by a party to participate in the FP Application Infrastructure platform, as EnergyMarketParticipant, ThirdPartyInformationProvider, DSO or Supplier. To be deployed by the party. Typically provides access to the external communication interfaces of the MC and Nodes. Toolbox offered by the store to lower the barrier of entering the FP Application Infrastructure platform.
	Information application	Information application for parties, such as EnergyMarketParticipants, to help them with analysis/visualization of their interactions with an FP Net.
Licenses	Third Party Information Provider License	A renewal of a license, previously purchased together with an App that works with its server-side information provision component, or separate license for use of a third party information provider that supplies information for standardized information feeds.

4.3.3 *Other possible offerings from an FP AppStore*

A store could offer combinations of FP Node Apps or AppGroups with other applications or tools from the previous paragraph. It could even make offerings involving energy consumption devices. It could even make combinations involving specific energy market participants. An example could be to offer devices from a certain brand or set of capabilities together with an AppGroup, consisting of their ResourceManager, ResourceDriver, RM UI App and Energy App, to work with a specific EnergyMarketParticipant. One could of course also offer “embedded resources”, which are appliances that have ResourceManager and ResourceDriver functionality embedded. The possible offerings are subject of further design discussions.

4.3.4 *Functionalities of Apps and examples*

We review different examples of App functionalities and elaborate which App category is needed to accomplish this and how the App interacts with the FP Node and other parties in an FP Net

[49] This is the case for example for the PowerMatcher App that communicates (via the PMBids interface specification) with its server-side PowerMatcher Auctioneer in the energy market.

4.3.4.1 *GUI app to view history graphs of consumption of devices*

To realize a GUI that shows history graphs of the consumption of Devices, this can be realized as GUI Widget app, accessing the SMF Model to collect data and render it.

4.3.4.2 *GUI app to view/control specifics of a device*

To realize a GUI for viewing and controlling specifics of a Device, this can be realized as a Resource Manager UI App working together with the ResourceManager app.

4.3.4.3 *GUI app to access FP AppStore*

A GUI to have a rich user experience for visiting and shopping in the FP AppStore can be realized as a Store UI app. Such an App is typically provided by the store itself, offering a free purchase to replace the default shopping-interface provided by the node software.

4.3.4.4 *GUI app providing energy view information via walkthrough*

When one wants to provide a GUI that shows energy consumption/generation information of Devices based on the location and orientation of the UserDevice, it can be realized by feeding the sensor data (location, orientation, gyroscope) from its UserDevice to the Node, a GUI Widget App can use this data to show the energy information of the Device. To show Device information based on the location and orientation of the UserDevice, the GUI could present a floor plan to support the walkthrough experience. It is even possible to overlay the energy information of the Devices on the camera image to create a virtual reality view. This is possible with a non-FP Node App deployed on the User Device (e.g. iOS or Android App) where the App retrieves the necessary Device information from the Node using its WebServices. To store the location of Devices in the Node, a GUI Widget App could help with the entry of this location data, the information itself can be preserved using the CoreStorage facility.

4.3.4.5 *QR-code scanning applications*

An application to scan the QR-code of a device to view its energy related information could be realized in a number of ways. A non-FP Node app (iOS app for example) can send the QR-code scan to a server-part that feeds the extracted device identification back to the node (via a DestinationIdentification or InformationFeed labeled information third-party information package).

Using QR-codes to help with the installation of new devices can be of particular interest to ease the process of installation of Devices. A Store UI app that provides a rich store experience can use the QR-code-scan provided as sensor-data to the Node, send it to the store for identification and adapt the store GUI to present the appropriate AppGroup to install the device easily. This AppGroup typically has a ResourceManager and a ResourceDriver, but could also include Resource Manager Helper apps and a Resource Manager UI app.

Providing QR-code stickers on home appliances could be a good practice for manufacturers or stores to ease the installation process of their Devices.

4.3.4.6 *Geo-fencing and other location related applications*

Location data of the User Device, delivered to the Node as sensor data, can help the GUI. This can help the GUI to focus on Devices on the current location in a multi-location account. It can also help the GUI to focus when setting dynamic energy targets for Devices since it can present the Device choices based on their proximity to the UserDevice.

One could implement geo-fencing applications to take the location of a UserDevice into account to influence energy target settings. Besides a geo-fencing application for a single UserDevice, using the location information from the set of UserDevices (such a smartphones) for all members of a

household can be particularly powerful to influence the energy target settings. Based on the distance of all members of the household to the account premises, the energy target settings could be changed.

4.3.4.7 *Information provision to influence energy management*

Information provision such as weather information/forecasts can improve the coordination by the Energy App. An Energy App Helper can receive third party information from its server-side part about weather information, or it can receive weather forecasts via a standardized InformationFeed from other providers. A Resource Manager Helper App could exploit weather information in a similar way.

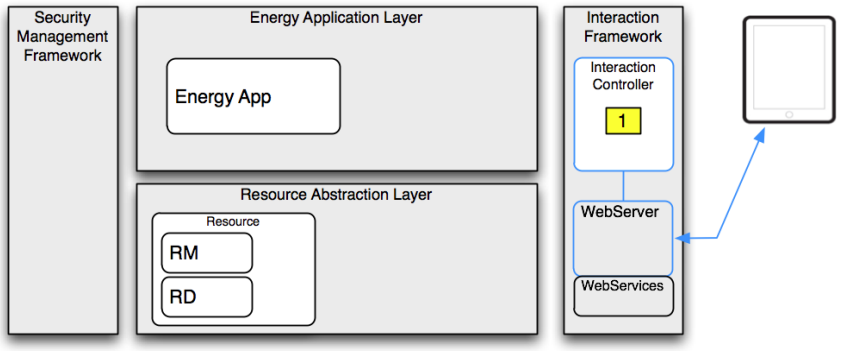
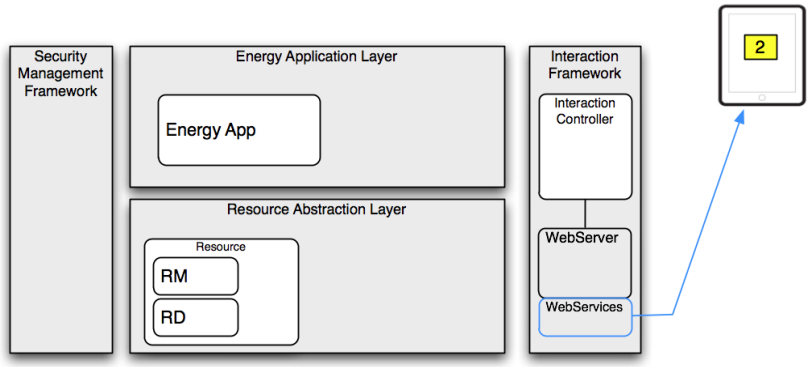
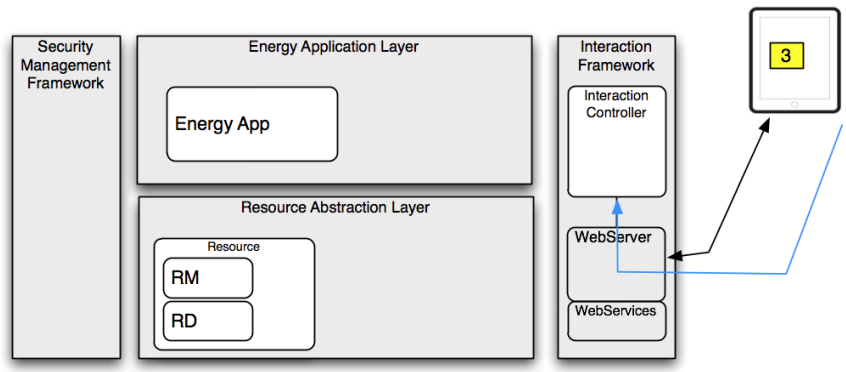
Instead of general weather information and forecasts, it may be useful to have ambient condition information such as the current home or outside temperature. Note that the EnergyServices external communication interface not only provides total energy consumption and generation info, but in general support reading (smart) meter information. This information, in this example from a (smart) meter thermometer, is made available in the SMF model where it can be accessed by decision logic components and Apps in general.

4.3.4.8 *Model information*

As already discussed in the paragraph on the Analysis Center of an FP ManagementCenter, server-side parts of Apps or ThirdPartyInformationProviders in general can provide useful services by performing analytics/learning/forecasting based on the historical data available in the FP ManagementCenter. These services can involve data of the account only, or even involve data of a whole group of similar accounts or all accounts of the FP Net. Matching the account profile with others to derive improvements of energy settings or forecasts seems a very promising approach. This matching process is something that is useful anyhow for an FP ManagementCenter to determine the EnergyType of an account upon its first installation. This to make sure that it can already start with a good energy profile and set of static energy targets efficient for the profile of the account.

4.3.5 *Interaction patterns of Apps*

The next table illustrates typical interaction patterns of Apps, which Apps can combine of course.

Interaction Pattern of Apps	*Description*
 Figure 4-74 Interaction pattern of Apps – GUI App	A GUI App (1) enriches the GUI as consumed on a User Device.
 Figure 4-75 Interaction pattern of Apps - WebServices	A non-FP Node client app (2) such as an iOS app, an Android app or PC application provides a GUI by interacting via the WebServices of the Node
 Figure 4-76 Interaction pattern of Apps – Sensor Data enriching GUI	Feeding sensor data from the UserDevice (3) to the Node to enrich the GUI, consuming/providing the GUI as either of the previous patterns (1) or (2).

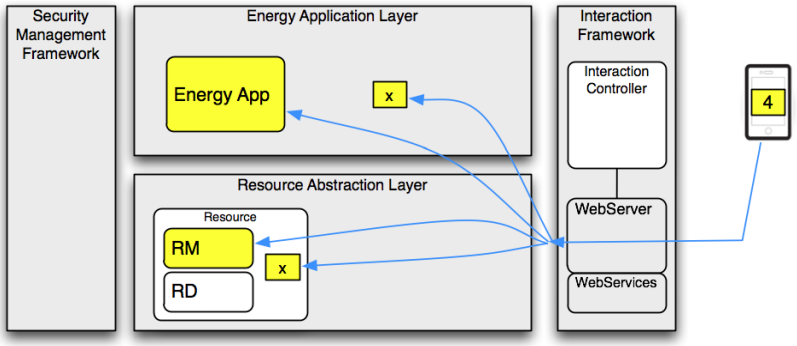


Figure 4-77 Interaction pattern of Apps – Sensor Data to others

Feeding sensor-data from a UserDevice (4) into the Node to help the decision logic; via a ResourceManager App, Energy App, or their helper Apps.

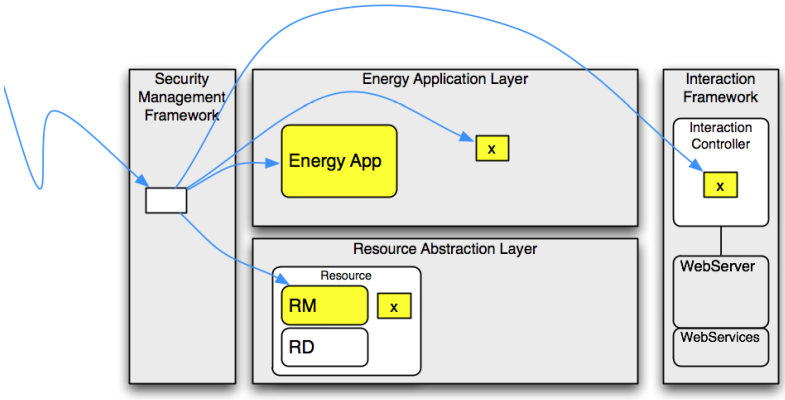


Figure 4-78 Interaction pattern of Apps – Third party information

Apps receiving third party information from their server-part or from a general provider of a standardized feed, helping decision logic or enriching the Gui.

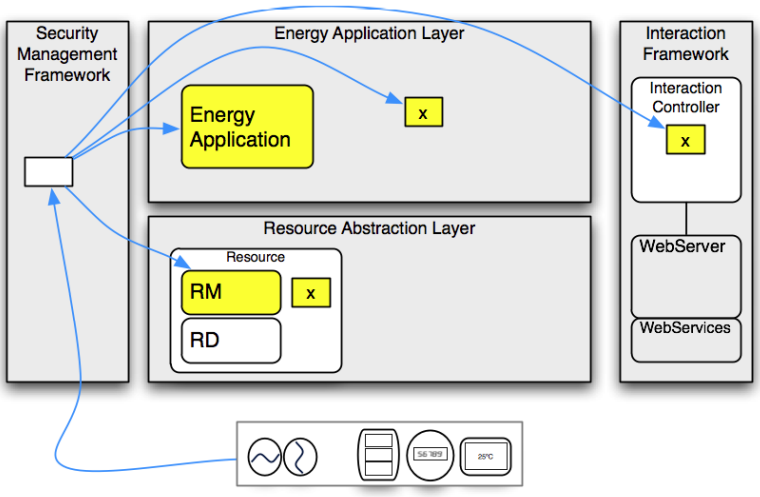


Figure 4-79 Interaction pattern of Apps – Using meter data

Apps using smart-meter sensor information, such as total consumption/generation, ambient condition sensors for home and outside temperature, etc. Remark that Apps can use this information from accessing the SMF Model, since the information is provided to the SMF Model via the Es interface.

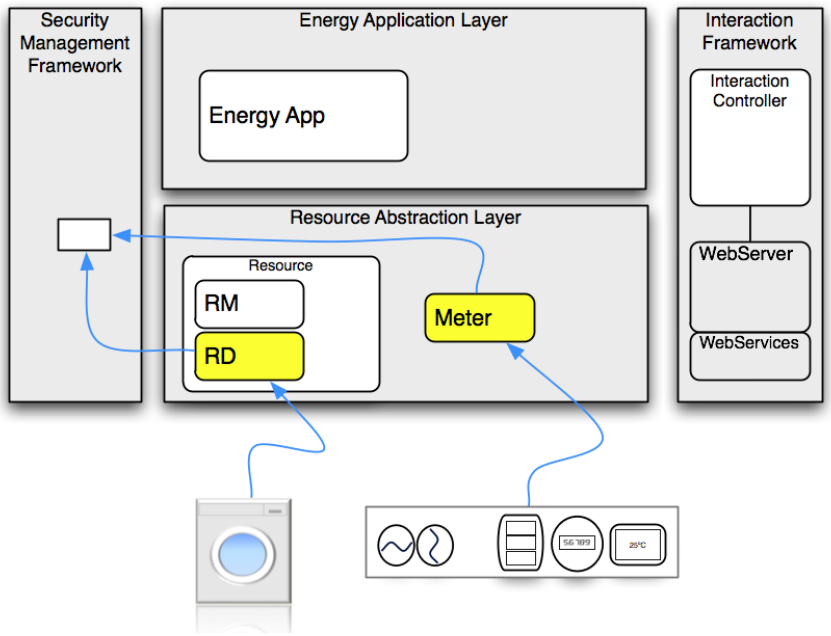


Figure 4-80 Interaction pattern of Apps – Providing meter data

Apps providing smart-meter sensor information, such as total consumption/generation, ambient condition sensors for home and outside temperature, etc.

If an appliances provides metering information, the Device Driver App can provide it to the framework. Otherwise a Meter App can provide it from smart meter plugs.

5 Appendix – FlexiblePower Runtime Application Interface

5.1 Introduction

This document describes the FlexiblePower Resource Abstraction Interface (FP RAI)

This interface is being used to express the electrical flexibility that appliances have to offer and how this flexibility should be exploited.

The main concepts are the control space, which is used to describe energetic flexibility of appliances, and the allocation, which indicates how this flexibility should be used.

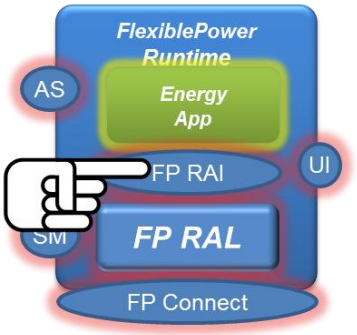


Figure 105: Overview of the FlexiblePower Runtime

The control space and allocation concepts are the subject of paragraph 5.2. The chapters that follow describe different specializations (timeshifter, buffer, storage and uncontrolled load/generation) of the control space concept.

5.2 Control Space and Allocation

This paragraph describes the ControlSpace parent class from which more specific ControlSpaces are derived. A ControlSpace is used to convey the energetic (more specifically electrical) flexibility of a particular resource/appliance.

It is the Resource Manager that constructs and communicates a ControlSpace. In response to a ControlSpace a Resource Manager may receive an Allocation. The Allocation describes how the flexibility within a ControlSpace should be used. It contains a precise energy profile (within the boundaries of the indicated flexibility) that the Resource Manager should try to follow as closely as possible.

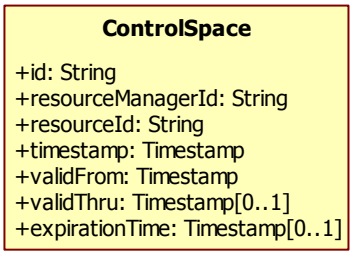


Figure 82: ControlSpace Class

ControlSPace		
Description:	This is a parent class from which more specific `ControlSpace` classes can be derived.	
Attributes:	`id`	An identifier that uniquely identifies this `ControlSpace`.
	`resourceManagerId`	An identifier that uniquely identifies the Resource Manager that produced this `ControlSpace`.
	`resourceId`	An identifier that uniquely identifies the resource or appliance that this Control Space is about
	`timestamp`	The moment in time this `ControlSpace` was created.
	`validFrom`	A timestamp that indicates from which moment on this `ControlSpace` is valid. In other words: the start time of the flexibility.
	`validThru`	An optional timestamp that indicates untill which moment this `ControlSpace` is valid. In other words: the end time of the flexibility.
	`expirationTime`	An optional timestamp that indicates the moment in time this `ControlSpace` expires. An `Allocation` that is sent in response to this `ControlSpace` should be received before the expirationTime. The expirationTime will typically lay before the validThru timestamp and will be used when it takes time to process an `Allocation`.

Figure 83: Timestamp helper class for ControlSpace

Timestamp		
Description:	This class is used to indicate a moment in time. The `ControlSpace` related classes use their own version of a `Timestamp` (instead of a version that is defined in an external library), so that they are independent of additional libraries.	
Attributes:	`timestampInMilliSeconds`	This attribute indicates the number of milliseconds that elapsed since the 1st of January 1970 in Coordinated Universal Time (UTC).

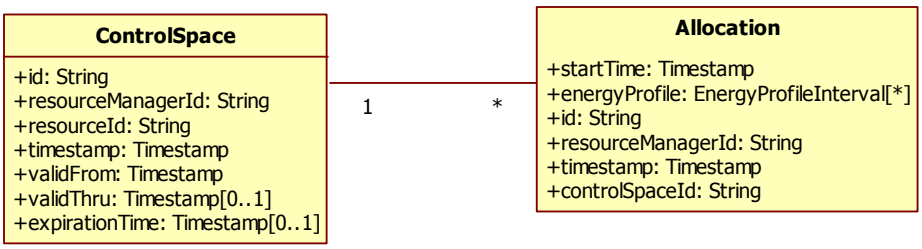


Figure 84: Allocation class in relation to ControlSpace

Allocation		
Description:	An Allocation class is a reaction to a ControlSpace and indicates how the flexibility in a ControlSpace should be used. It contains an energy profile that should be adhered to by the Resource Manager that sent the ControlSpace. A ControlSpace can be associated to zero or more Allocation objects. Multiple Allocation object are only applicable when an update of an Allocation is used to overrule the original/previous Allocation.	
Attributes:	startTime	This timestamp marks the start of the energy profile that is expressed in the energyProfile attribute.
	energyProfile	This attribute describes the energy profile which can be visualized as a bar graph where the height of each bar indicates the amount of energy that is consumed or generated. The width of a bar indicates the duration. The energyProfile attribute consists of an array of EnergyProfileInterval objects.
	id	An identifier that uniquely identifies this Allocation.
	resourceManagerId	An identifier that refers to the Resource Manager that is to receive this Allocation.
	timestamp	The moment in time this Allocation was created.
	ControlSpaceId	An identifier that refers to the ControlSpace that this Allocation is a response to.

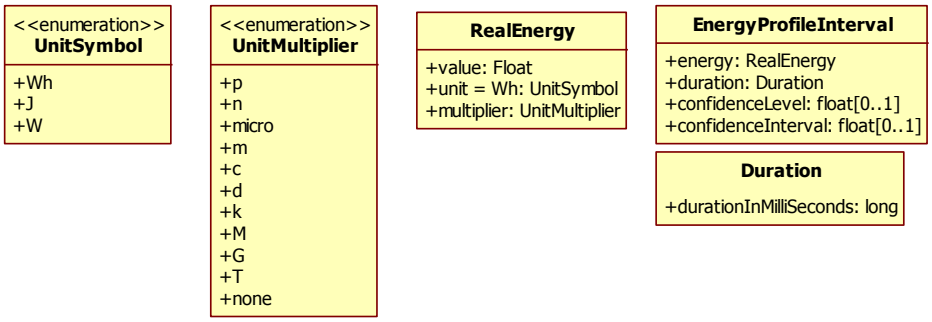


Figure 85: Allocation helper class

EnergyProfileInterval		
Description:	This class describes a single interval of an energy profile.	
Attributes:	`energy`	This attribute describes the amount of energy within this interval. Consumption is indicated by a positive number, generation by a negative number. The class of the energy attribute is `RealEnergy`.
	`duration`	This attribute indicates the duration of this interval and is of the type `Duration`.
	`confidenceLevel`	This is an optional attribute which can be used to express uncertainty about the `EnergyProfileInterval`. It is always used in combination with the confidence interval upper – and lower boundary attributes. The confidence level is a percentage that indicates how confident one is that the true amount of energy will lay between the boundaries of the confidence interval. Typical values for the confidence level are 95% and 99%
	`confidenceIntervalUpperBoundary`	This optional attribute is always used in combination with the confidence level and lower boundary attributes. It indicates the upper boundary of the confidence interval and should always be greater than the value of the energy attribute.
	`confidenceIntervalLowerBoundary`	This optional attribute is always used in combination with the confidence level and upper boundary attributes. It indicates the lower boundary of the confidence interval and should always be smaller than the value of the energy

		attribute.

Duration		
Description:	This class is used to express duration. The `ControlSpace` related classes use their own version of a `Duration` (instead of a version that is defined in an external library), so that they are independent of additional libraries.	
Attributes:	`durationInMilliSeconds`	This attribute expresses the duration in milliseconds.

RealEnergy		
Description:	This class was copied from the Common Information Model (CIM) by the International Electrotechnical Commission (IEC) [CIM09]. Where possible international standards are reused as much as possible.	
Attributes:	`value`	This attribute is a Float which, together with the multiplier attribute, expresses the amount of energy.
	`unit = Wh`	This attribute is of the enumeration type `UnitSymbol`, which contains multiple symbols. Out of these symbols `Wh` is to be used within the `RealEnergy` class.
	`multiplier`	This multiplier attribute (of the enumeration type `UnitMultiplier`) can be added to the unit to form constructs such as `kWh` or `MWh`.

UnitSymbol	
Description:	This enumeration was also taken from [CIM09]. It contains symbols for energetic units.

UnitMultiplier	
Description:	This enumeration was also taken from [CIM09]. It contains multipliers that can be combined with the `UnitSymbol` enumeration.

5.3 Timeshifter Control Space

Timeshifters are a category of appliances that produce or consume energy according to a predetermined energy profile. The flexibility of timeshifters comes from their ability to shift the start time of this profile.

Typical examples of timeshifting appliances are washing machines and dish washers. The control space of a timeshifter is described below.

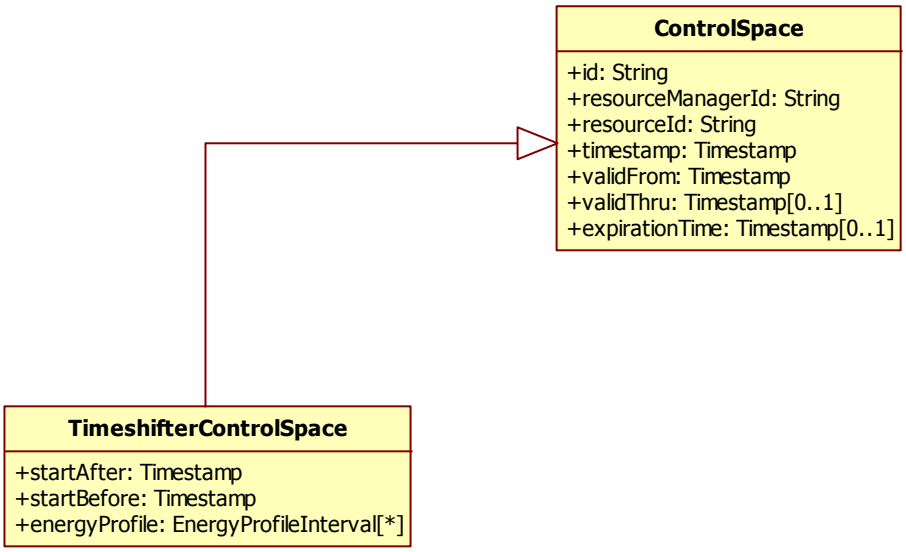


Figure 86: TimeshifterControlSpace class

TimeshifterControlSpace		
Description:	This class is a child class of `ControlSpace` that is used to describe the flexibility that is offered by timeshifters. The timeshifter can be started within a certain period of time.	
Attributes:	`startAfter`	This attribute is of the type `Timestamp` and indicates the beginning of the timeshift period. In other words the start time should be after this point in time.
	`startBefore`	This attribute is also of the type `Timestamp` and indicaties the end of the timeshift period. In other words the start time should be before this point in time.
	`energyProfile`	This attribute describes the actual energy profile of which the start time can be shifted. This profile can be visualized as a bar graph where the height of each bar indicates the amount of energy that is consumed or generated. The width of a bar indicates the duration. The energyProfile attribute consists of an array of `EnergyProfileInterval` objects.

5.4 Buffer Control Space

Buffers are a second category of appliances that can provide electrical flexibility. With a buffer appliance one can choose to consume/produce more energy now (within certain operational constraints) so that it consumes/produces less energy later or the other way around.

Most buffers will be thermal. Examples of such appliances are fridges, freezers and heating systems. The control space of a buffer is described below.

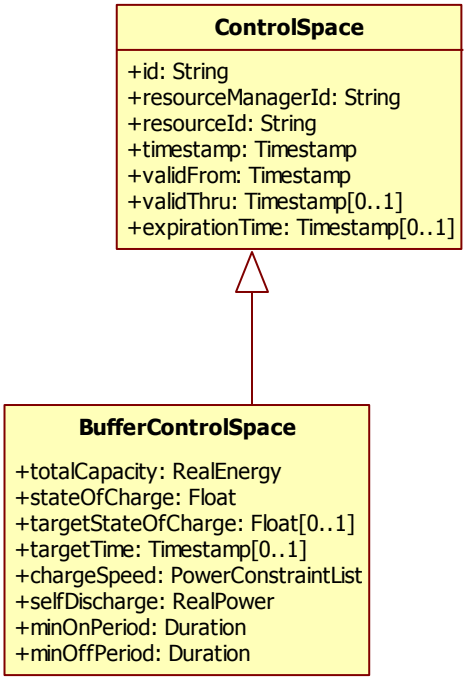


Figure 87: BufferControlSpace class

BufferControlSpace		
Description:	This class is a child class of `ControlSpace` that is used to describe the flexibility that is offered by buffers.	
Attributes:	`totalCapacity`	This attribute is of the type `RealEnergy` and expresses the total amount of electrical energy that this buffer can absorb (positive value) or generate (e.g. in case of a micro CHP; negative value). The total capacity of a freezer for instance equals the amount of energy that is needed to cool a freezer from the highest acceptable temperature (e.g. -18˚C) to the lowest acceptable temperature (e.g. -24˚C). The temperature boundaries can be configured by the end user via the Resource Manager
	`stateOfCharge`	This attribute is of the type Float and is a percentage that indicates how much of the totalCapacity is used. In the case of the freezer a state of charge of 0% means that the freezer can still absorb its total capacity. When the lowest acceptable temperature is reached it is not able to absorb any more energy which equals to a state of charge of 100%.
	`targetStateOfCharge`	This optional attribute can be used to specify a target percentage that should be reached by a certain point in time. For example if a certain amount of tap water is required at 7 a.m. for showering this can be expressed using the target state of charge. This attribute is always used in combination with the targetTime attribute.
	`targetTime`	This optional attribute is of the type `Timestamp` and is always used in conjunction with the targetStateOfCharge attribute. It specifies the point in time when the target state of charge should be reached.
	`chargeSpeed`	The charge speed is the power that the buffer draws from the socket (in the case of consumption this will be a positive value) or delivers back to the grid (negative value). The charge speed may consist of a single value, multiple discrete values and/or continuums (needed to describe modulating appliances). This can be expressed in the `PowerConstraintList` class, which is the type of this attribute.

	`selfDischarge`	Buffers are typically not perfectly isolated. This means that energy will leak away. This is being expressed through this attribute. It is of the type `RealPower`.
	`minOnPeriod`	Many appliances cannot be switched on and off in quick succession as this would wear them down prematurely. This attribute, of the type `Duration`, is used to specify a minimum period for which this appliance has to be switched on.
	`minOffPeriod`	Many appliances cannot be switched on and off in quick succession as this would wear them down prematurely. This attribute, of the type `Duration`, is used to specify a minimum period for which this appliance has to be switched off.

-

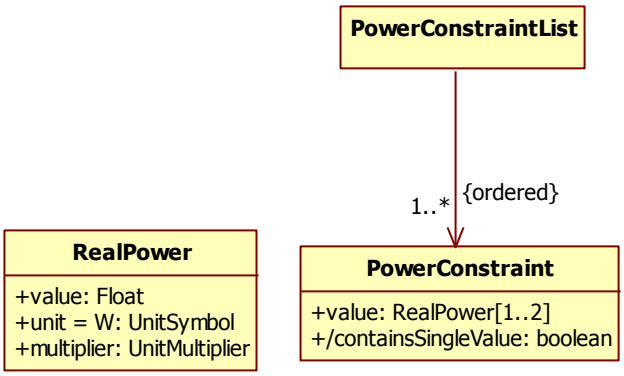


Figure 88: Helper classes for BufferControlSpace class

`PowerConstraintList`	
Description:	Instances of this class aggregate a list of `PowerConstraint` instances. A `PowerConstraint` is either a fixed value or an upper and lower bound. With this list structure combinations can be made, e.g.: *1 kW* *1 kW* or *2 kW* or *3 kW* between *1* and *3 kW* *1 kW* or *2* between *3 kW*

PowerConstraint		
Description:	Describes a constraint on an amount of power consumed or provided. If one value is provided this value is the only allowable value (unless contained in a `PowerConstraintList` instance with multiple elements). If two values are provided, this indicates an upper and lower bound.	
Attributes:	`value`	The values as described above.
	`/containsSingleValue`	This boolean is a derived attribute that indicates whether a single value is provided (true) or whether 2 values have been provided (false) indicating a lower and upper bound.
Constraints:	`self.value[0] < self.value[1]` In case two values have been provided (referring to a lower and an upper bound) the above mentioned constraint applies. It specifies that the first value provided should be smaller (lower bound) than the second value (upper bound).	

RealPower		
Description:	This class was copied from the Common Information Model (CIM) by the International Electrotechnical Commission (IEC) [CIM09]. Where possible international standards are reused as much as possible.	
Attributes:	`value`	This attribute is a Float which, together with the multiplier attribute, expresses the amount of power.
	`unit = W`	This attribute is of the enumeration type `UnitSymbol`, which contains multiple symbols. Out of these symbols `W` is to be used within the `RealPower` class.
	`multiplier`	This multiplier attribute (of the enumeration type `UnitMultiplier`) can be added to the unit to form constructs such as `kW` or `MW`.

5.5 Storage Control Space

The flexibility of the storage category of appliances has a lot of similarities to the flexibility that can be offered by buffers. The main difference between buffers and storage is that with buffers the electrical energy only flows in one direction: it is either consumption or generation. With a storage appliance the electrical energy flow is bidirectional.

A battery (management system) is a typical example of an appliance in the storage category. The battery may be autonomous, but it may also be part of an electrical vehicle for instance. The storage control space is described below.

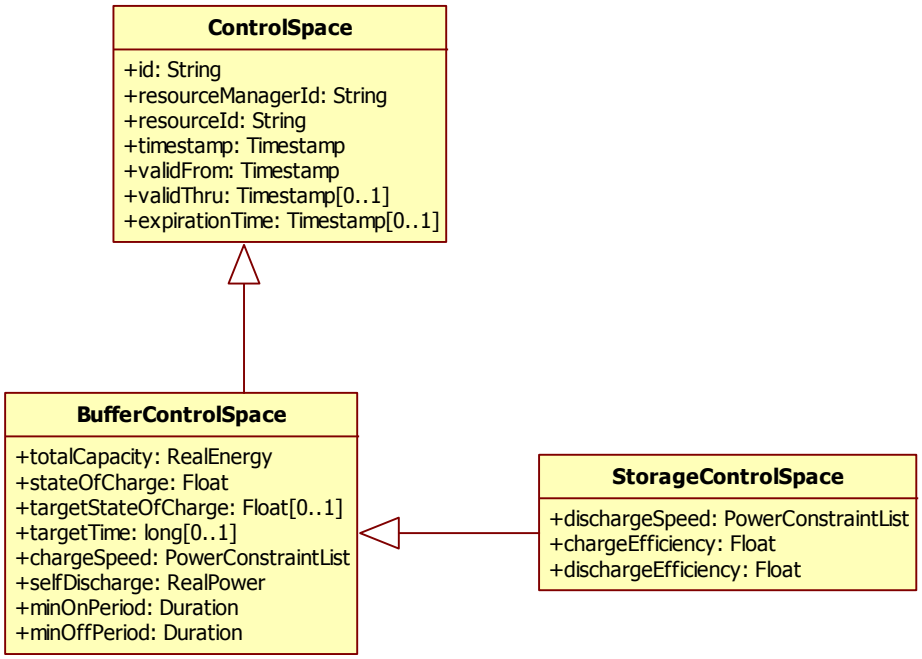


Figure 89: StorageControlSpace class

StorageControlSpace		
Description:	This is a child class of the `BufferControlSpace` and describes the flexibility that a storage appliance has to offer. It shares all `BufferControlSpace` attributes but also has some additional attributes that are specific for electricity storage.	
Attributes:	`dischargeSpeed`	This attribute specifies the power level(s) with which a storage appliance can deliver electricity back to the grid (this is regarded as production, therefore these will be negative value(s)). Multiple power output levels can be specified through the `PowerConstraintList` class which is the type of this attribute.
	`chargeEfficiency`	Not all the power that is drawn from the socket will end up being used to charge the storage appliance as there will be some losses in the charging process. This attribute is a Float which can have a value between 0 and 1 to indicate the efficiency of the charging process.
	`dischargeEfficiency`	There will also be losses during the process of discharging. This Float attribute can take on any value between 0 and 1 to indicate the efficiency of the discharging process.

5.6 Uncontrolled load/generation Control Space

The last category is the uncontrolled load/generation. This category is reserved for appliances that cannot be controlled. As a consequence they cannot offer flexibility. It is however important to know how these appliances will behave energetically, because this needs to be taken into account to make informed decisions about the usage of flexibility in the other three categories.

A PV panel is a good example of an uncontrolled generation appliance, whereas lighting represents an uncontrolled load. The uncontrolled load/generation control space is described below.

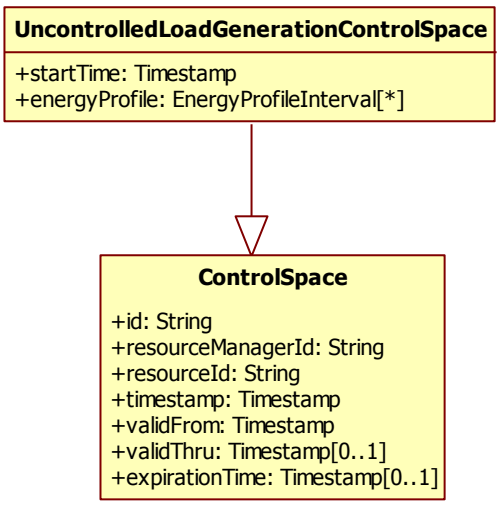


Figure 90: UncontrolledLoadGenerationControlSpace class

UncontrolledLoadGenerationControlSpace		
Description:	This is a child class of the `ControlSpace` and describes the expected consumption (load) or generation of uncontrollable appliances such as PV panels, lighting, etc..	
Attributes:	`startTime`	This is an attribute of the type Timestamp and marks the start of the energy profile that an uncontrolled appliance is expected to follow.
	`energyProfile`	This attribute describes the predicted energy profile. This profile can be visualized as a bar graph where the height of each bar indicates the amount of energy that is consumed or generated. The width of a bar indicates the duration. The energyProfile attribute consists of an array of `EnergyProfileInterval` objects. The reliability of the prediction can be expressed by using the confidenceLevel and confidenceInterval attributes of the `EnergyProfileInterval` class.

5.7 Overview Control Space diagram

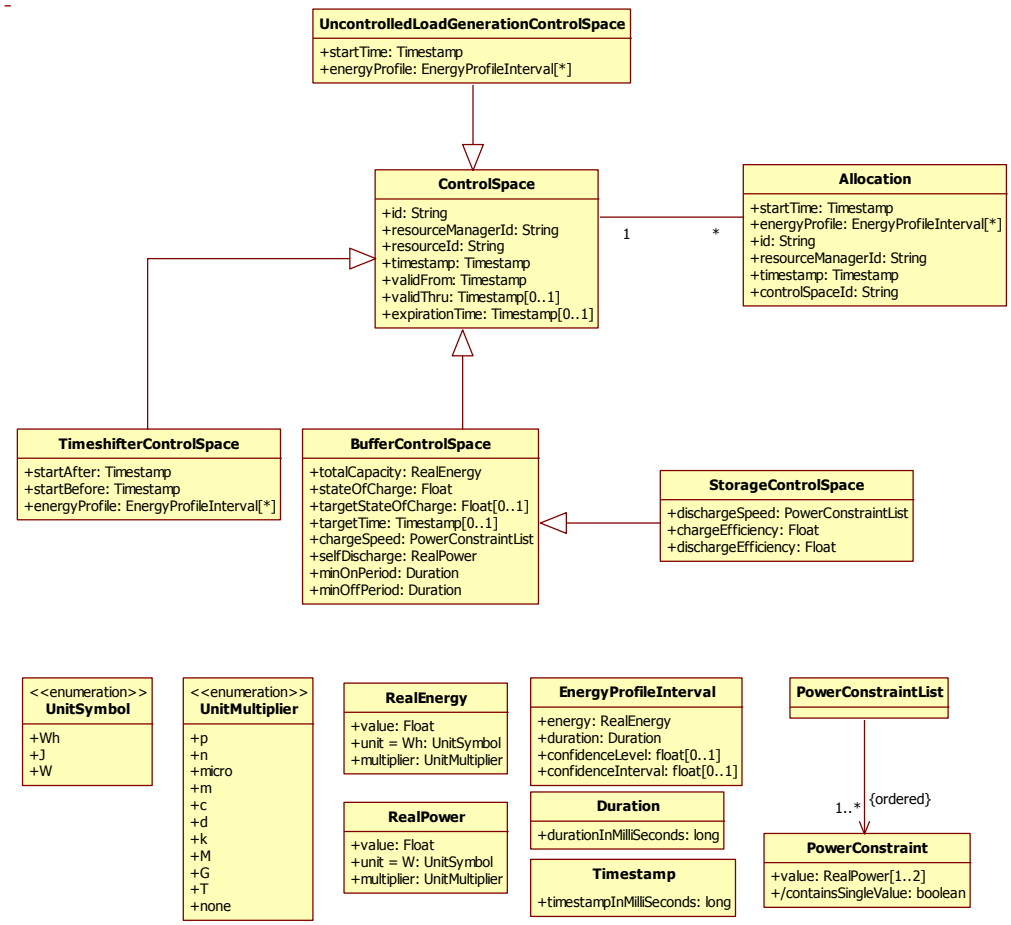


Figure 91: The complete control space class diagram

5.8 References

[CIM09] IEC61970-301 Ed. 2, 'Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM) base', International Electrotechnical Commission, 2009