

Chapter 9

The IoT ARM Reference Manual

Martin Bauer, Nicola Bui, Christine Jardak, and Andreas Nettsträter

Whereas we explained the process of creating an IoT architecture with the support of the IoT ARM in Chap. 6 [Process Chapter] and gave an example how a concrete architecture can be defined based on different models and views of the IoT ARM in Chap. 11 [Concrete Architecture Chapter], we now provide reference manuals with guidelines how to use the IoT Domain Model, the IoT Information Model, the IoT Communication Model and the Perspectives when creating a concrete architecture.

Starting with the IoT Domain Model.

9.1 Usage of the IoT Domain Model

This section is intended for architects who want to apply the IoT Domain Model on a specific use case. We discuss typical instantiations of the IoT Domain Model. These model cases can be used as basic patterns when doing concrete modelling.

M. Bauer (✉)

NEC Laboratories Europe, Software & Services Research Division, NEC Europe Ltd,
Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
e-mail: Martin.Bauer@neclab.eu; www.nw.neclab.eu

N. Bui

Consorzio Ferrara Ricerche, Via Savonarola 9, 44122 Ferrara, Italy
e-mail: buincl@unife.it; www.unife.it

C. Jardak

Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany
e-mail: christine.jardak@siemens.com; www.siemens.com

A. Nettsträter

Fraunhofer Institute for Material Flow and Logistics IML, Joseph-von-Fraunhofer Str. 2-4,
44227 Dortmund, Germany
e-mail: andreas.nettstraetter@iml.fraunhofer.de; www.iml.fraunhofer.de

9.1.1 Identification of Main Concept Instances

Similar to the identification of stakeholders and actors in standard software engineering practices, the IoT Domain Model is used in a first step of the architectural design process in order to:

1. Identify Physical Entities and related Virtual Entities;
2. Identify Resources (at least from a functionality perspective);
3. Identify Devices (or device options);
4. Identify Services;
5. Identify Users.

The identification of Resources and Devices is used together with the IoT Communication Model to define the communication paradigms and how these devices and resources interact. This is comparable to interaction models in standard software engineering practices. The Services to be used and where they should be deployed are analysed and finally the Users of these services are identified.

9.1.2 Modelling of Non-IoT-Specific Aspects

It is important to understand that the IoT Domain Model is not attempting to be a domain model for all types of ICT systems. Rather, it focuses on the IoT-specific parts. When modelling a complete system, many of the aspects to be covered are not IoT-specific. For these aspects, the IoT Domain Model will provide only little help.

For example, the Service concept in the Domain Model is primarily focused on modelling IoT Services that directly or indirectly expose Resources; however, the Service concept also can be used to provide a link to general services in the ICT domain.

9.1.3 Identifiers and Addresses

Identifiers and addresses are logically two different concepts, which unfortunately however are often confused in practice, in particular in the discussions about IoT (Haller 2010). While in some cases the address might be used in the role of an identifier, it is important to distinguish between these terms.

Identifiers are used to identify something, for example a Physical Entity. In this case, the identifier is an attribute of the related Virtual Entity. Examples include URIs (Uniform Resource Identifiers as used on the Web, e.g. `foo://example.com/building1/room3`), EPCs (Electronic Product Codes, e.g. `01.23G3D00.8886A3.365000A03`)

(EPC Tag Data Standard) and uIDs (uCode Identifiers,¹ e.g. 0123456789ABCDEF0123456789ABCDEF).

Addresses, on the other hand, are means for locating, accessing or communication with something, e.g., a service or a device. Addresses manifest themselves as attributes of the corresponding concepts, i.e., attributes of a service or a device. Examples include IPv6 or MAC addresses.

As mentioned above, there are cases in which it can make sense to use addresses as identifiers, e.g. when the address uniquely identifies the Physical Entity. For example, a street address is good identifier for a building, but not for a human being. An e-mail address on the other hand provides a unique way of identifying people.

Modelling Option 1	An address can be used as an identifier for a Physical Entity (and the corresponding Virtual Entity) if it uniquely identifies it.
---------------------------	--

Overall, identification and addressing are very important aspects of IoT systems. When designing an IoT system the different options should be evaluated and decided on early in the process, but as the decision depends on various requirements, assumptions and even technology choices, we cannot give specific recommendation on the reference model level.

9.1.4 Granularity of Concepts

In the IoT Domain Model, concepts like Device, Resource, and User have specialisations. Pertinent examples for Devices are Sensors and Actuators. When modelling a concrete scenario, one can use either the general concepts or their specialisations; the IoT Domain Model does not prescribe anything. For example, instead of using a concrete concept like Sensor it is also possible to use a more general concept like Device. However, the specialisations are more precise and are therefore preferable where they apply. In other words, if at the time of modelling it is not (yet) clear what type of device is used, then just use Device.

Modelling Rule 1	Model as precisely as possible based on the domain model concepts at the time of modelling. Use the more concrete, more fine-granular concepts and instances whenever possible, but only to the granularity that appears reasonable for the given purpose.
-------------------------	--

¹ <http://www.uidcenter.org/spec#UID-00010>.

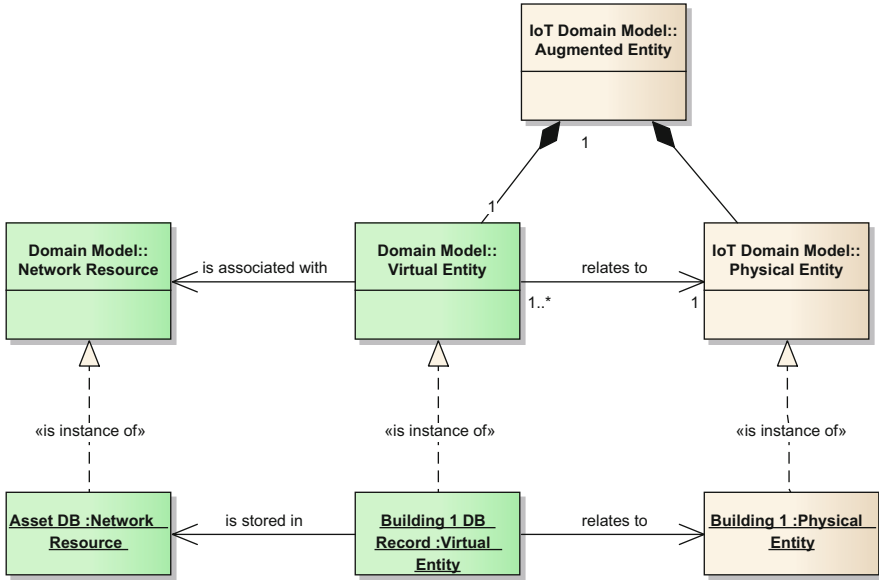


Fig. 9.1 Data-base pattern as an example for an augmented entity

9.1.5 Common Patterns

9.1.5.1 Augmented Entities

As described in Sect. 7.3.2.2, Augmented Entities are the composition of a Physical Entity with its related Virtual Entity. In many cases though, the Augmented Entity is of little practical relevance and will have no concrete instantiation, as the example in Fig. 9.1 shows. In this figure, a typical pattern is shown for how Physical Entities are mapped to data base records: In a data base of assets (a Network Resource in terms of the IoT Domain Model), a data base record (Virtual Entity, and also a Passive Digital Artefact) is stored for every building (Physical Entity).

Modelling Option 2	The Virtual Entity for a given Physical Entity can be a data base record stored in a Network Resource.
---------------------------	--

A different case is truly smart objects, i.e., intelligent devices that have embedded logic seemingly able to act autonomously. In this case, the Augmented Entity is the smart object itself, and the associated Virtual Entity is an Active Digital Artefact, namely, the embedded logic (e.g., the software agent).

Figure 9.2 shows an example of a smart object: an *Unmanned Aerial Vehicle* (UAV). The body of the UAV can be considered the Physical Entity, while the

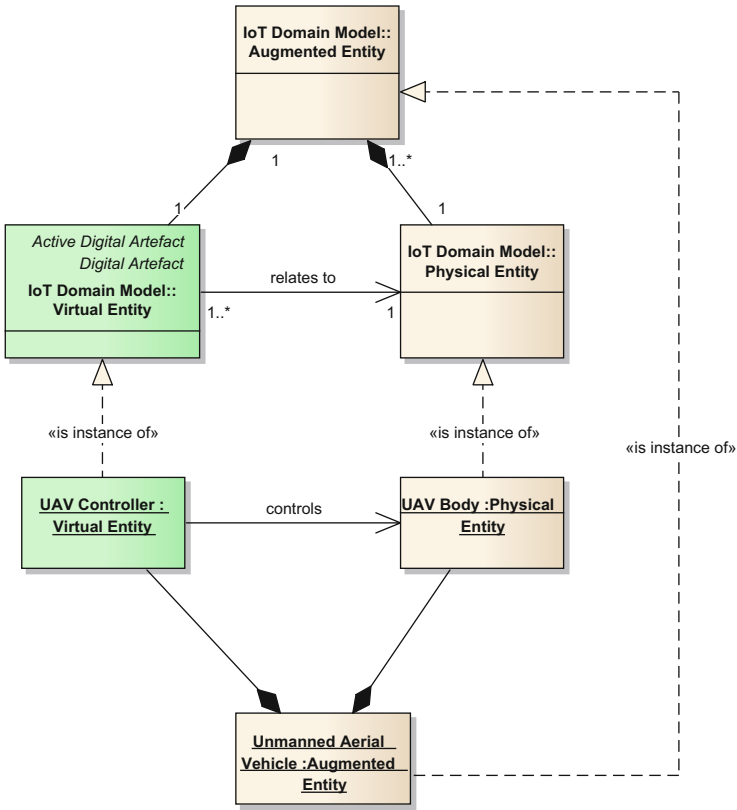


Fig. 9.2 Smart-object pattern. UAV: unmanned aerial vehicle

UAV controller is the related Virtual Entity. Together they form the Augmented Entity, the smart object.

Modelling Rule 2	When modelling an autonomous object, an Augmented Entity is used, consisting of a device (Physical Entity) and its software controller (Virtual Entity).
-------------------------	--

Finally, the question often arises if something should be modelled as a Physical Entity or not. While possibly every real-world object could be modelled as a Physical Entity, this does not make sense. Not every sand corn needs to be represented in an IoT system. Hence we can deduce:

Modelling Rule 3	Only model something as a Physical Entity if it is relevant in the IoT system so that the representing Virtual Entity is also modelled.
-------------------------	---

9.1.5.2 Multiple Virtual Entities

In order to understand the case of multiple Virtual Entities, we take the example of a customer buying a new car. The customer visits the exhibition of an automobile manufacturing company and buys a new car. He then registers it under his name at the department of motor vehicles. In order to protect himself from unexpected financial expenses resulting from traffic collisions, he decides to buy a car insurance. In this small scenario we notice that the same car, which is the Physical Entity, is registered at three stakeholders: the manufacturer, the vehicle-registration department, and the insurance company. As depicted in Fig. 9.3 each of the three stakeholders maintains a unique entry in its database identifying the car. These entries are multiple Virtual Entities representing the same car.

In practice, the number of Virtual Entities depends on the systems and domains, where the Physical Entity is represented and of course also which stakeholders are involved. We note that the characteristics of the Physical Entity change and, therefore, many of the Virtual Entities need to be maintained and kept up-to-date. Notice that the IoT Domain Model does not explicitly spell out any requirements on the maintenance of single and multiple Virtual Entities.

9.1.5.3 Smart Phones and Other Mobile User Devices

Smart phones are a very common element in many IoT-related scenarios. They are on the one hand Devices containing a multitude of sensors, but they also host apps (Active Digital Artefacts), Services, and Resources. Figure 9.4 shows this in exemplary fashion: John's smart phone is used as a Device to track the location of John, its owner. The GPS sensor is embedded in the phone itself. It is thus embedded sensor hardware. Its data is made accessible through the related On-Device Resource and the location service that exposes it. An app can be used to display the location information.

Note that in this example (see Fig. 9.4), both the service as well as the application are shown to be hosted on the phone itself. While this depicts a common case, other instantiations are possible.

Instead of a smart phone other mobile user devices could be used, e.g. tablets or PDAs. The general modelling would be the same.

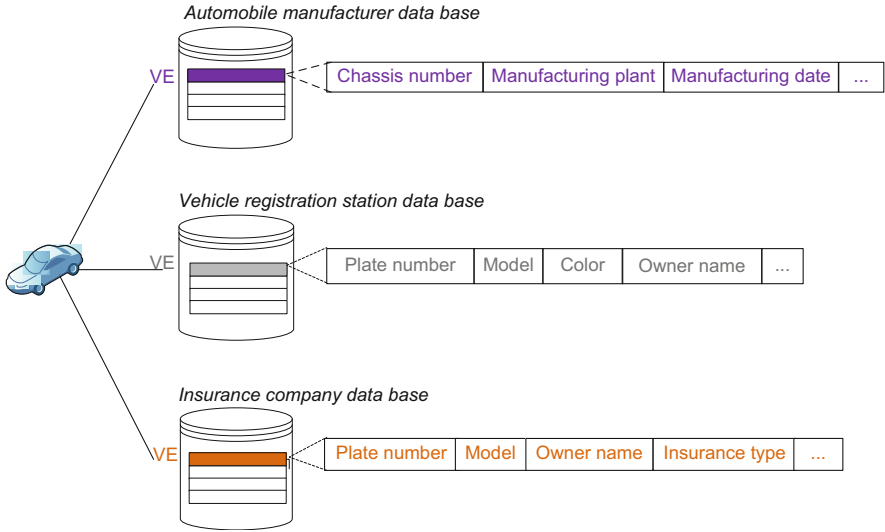


Fig. 9.3 Multiple virtual entities (data-base entries) for a single physical entity (car)

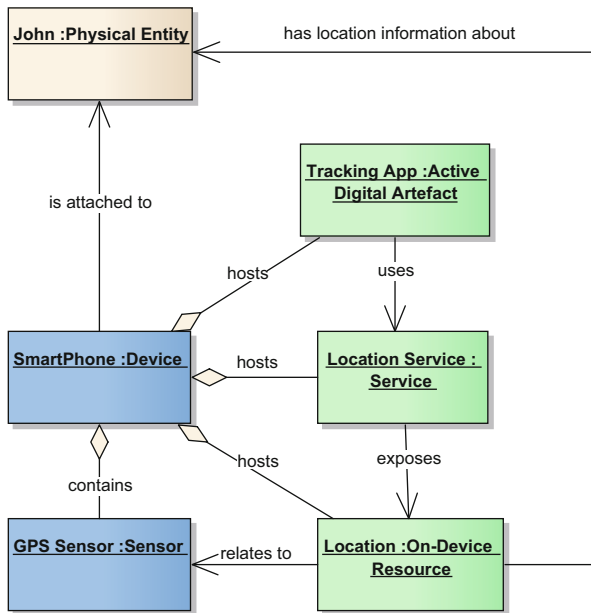


Fig. 9.4 Exemplary modelling of a smart phone that is used as tracking device

9.1.5.4 IoT Interactions

The IoT paradigm enables mediated interactions between Users and the physical world. This complements the direct interactions in the physical world that are

possible between Human Users and Physical Entities. It also enables the digital world, i.e. Active Digital Artefacts, to interact with the physical world.

9.1.5.5 Simple Mediated Interactions

A common case is that a User needs to access a Resource exposed through a Service in order to attain a given goal. Such goals may range from observing a Physical Entity by using a Sensor, to modifying its state by leveraging an Actuator device. We differentiate the following cases:

- **Retrieving information:** In this case a user would invoke a Service for retrieving some information. There are different options for the Service to get this information, which may be pull or push based. In case the Resource pushes the information, the Service would cache the information and provide it on request
- **Subscribing for information:** In the subscription case, the User subscribes to the Services and asynchronously receives notifications. After subscription, the Resource (e.g., on a Device) will detect the events of interest according to the specification provided by the user. The Service providing access to the Resource will then forward the event to the interested User. In an alternative implementation, the Service is performing the event detection by processing all the raw data from the Resource;
- **Actuation:** In the case, the User wants to control some aspect of the physical world mediated through the IoT system, it would call an Actuation service. In this case, the Service would interact with the Resource which would trigger the Actuator to execute the actuation.

9.1.5.6 M2M Interaction

Machine-to-Machine (M2M) communication is a technological approach for enabling meaningful information exchange between networked machines that show a certain degree of smartness. The term *machine* is generally related to an autonomous application while the smartness is related to the capability of controlling its own behaviour and communicating. This reflects the capability of making decisions on the basis of information retrieved from outside the system and being able to receive and execute commands. This approach is very relevant to the IoT and a specific definition of IoT Machine can be provided. In the terms of the IoT Domain Model, we define an IoT Machine as a composition of:

- An **Augmented Entity** whose Virtual Entity component is an Active Digital Artefact. In this way, it can start interactions (being a User, it can invoke Services) and can control the behaviour of the machine;

- One or more **Resources and the underlying Devices** which are used by the Active Digital Artefact to monitor/control the Physical Entity. Note that, because Resources are internal functionalities and the Active Digital Artefact is generally co-located on the same hardware, the interaction can happen even without the use of Services;
- The **Services** that are used for exposing Resources.

The example shown in Fig. 9.5 shows how a car interacts with a road barrier in order to speed up the passage through the barrier, i.e. that the barrier is removed as early as possible to enable the passage of the car. The incoming car is modelled as IoT Machine1, the automated barrier operator as IoT Machine2. The Machine1 Controller, an instantiation of an Active-Digital-Artefact Virtual-Entity, will access as a User (Active Digital Artefact can be Users) Service2 and will require the activation of the barrier. Service2 provides access to Functionality2 (Resource) related to Machine2 and thus, by accessing Service2, the car can retrieve the information about the barrier status which is needed in turn to decide whether it needs to slow down or can pass through without danger.

As M2M is about the communication-based interaction between machines, it is important to clarify that IoT Machines can also interact with non-IoT Machines. For example, an IoT-Machine could need certain information provided by an autonomous web application, a non-IoT Machine, in order to make decisions.

However, as the controlling program of Machine1 is a User according to the IoT Domain Model, it can also communicate with other Machines by calling appropriate embedded Services on another Machine, as shown in a simplified way in Fig. 9.6.

Object identification and tracking with RFID. The term “Internet of Things” was originally coined by the MIT Auto-ID Centre around 1999 (Ashton 2009), the precursor to what is now known as EPCglobal. EPCglobal is a standardization organization set up for achieving the worldwide adoption of the *Electronic Product Code* (EPC). It is based on RFID technology and the sharing of related information over the Internet. Due to its importance, it is worthwhile to map one of the most common scenarios of EPCglobal to the IoT Domain Model: the tracking of goods – pallets, cases, etc. – throughout the supply chain, from the manufacturer via distribution centres to the retail stores. A reverse mapping of EPCglobal onto the ARM can be found in Sect. 12.9.

A first thing to note is that we often have a hierarchy of Physical Entities and the related Virtual Entities. A large boxed pallet is identified by a shipping company as PE5 with its corresponding Virtual Entity VE5. As depicted in Fig. 9.7, the large boxed pallet contains multiple other cases that are identified as (PE1, VE1), (PE2, VE2), (PE3, VE3), and (PE4, VE4).

We note that the granularity of identifying PEs contained in other PEs is not defined by the IoT Domain Model, since it intimately depends on the application. In this example, if the large box contains four boxes of similar goods, e.g., shoes, the interest of the shipping company usually stops at identifying PE5 and thus tracking

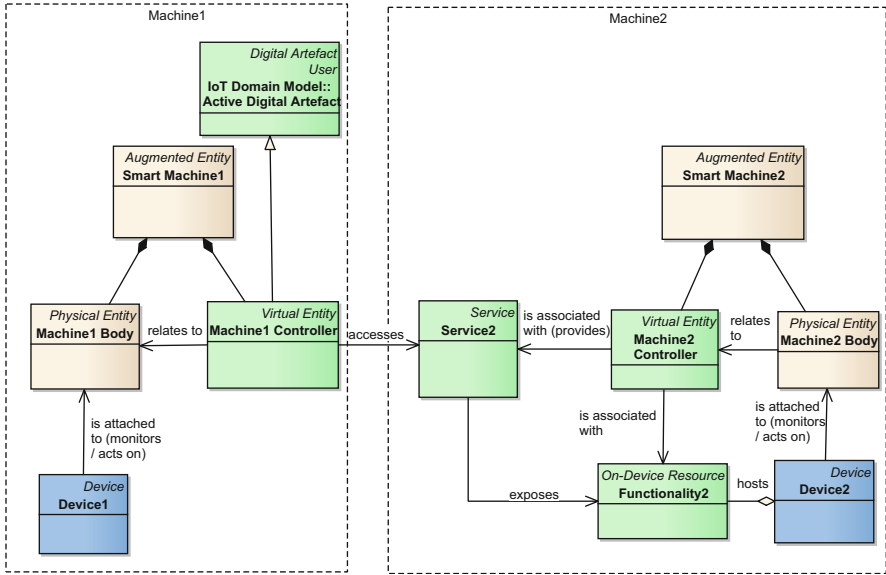


Fig. 9.5 IoT domain model instantiation for a M2M communication scenario



Fig. 9.6 M2M communication

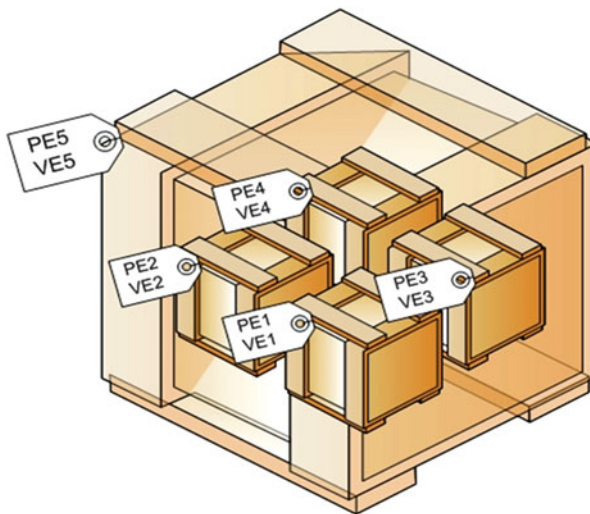


Fig. 9.7 Shipping box containing multiple packets. The VE-to-PE mapping is exemplified by paper tags

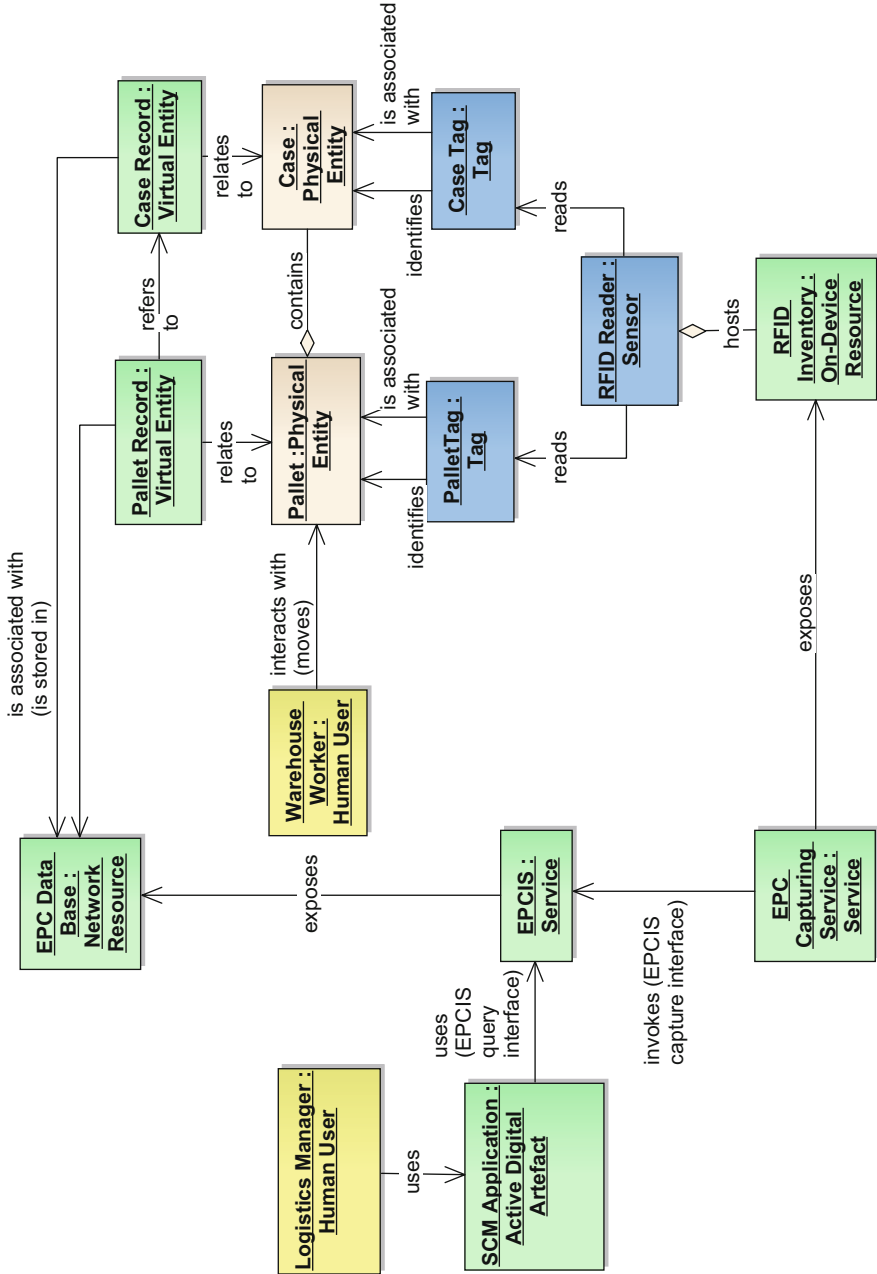


Fig. 9.8 Domain modelling of a typical EPC-based RFID scenario (pallet containing cases)

it by using VE_5 . Now if each of the four boxes contains different goods, e.g., shoes, hats, earrings, and bags, it might be of interest for the shipping company to additionally identify the four boxes as PE_1 , PE_2 , PE_3 , and PE_4 . The aim behind this higher granularity is to facilitate the process of sorting out the goods after delivery by checking VE_1 , VE_2 , VE_3 , and VE_4 .

The result of the whole mapping of the RFID logistics scenario, for only the pallet plus everything it contains, is depicted in Fig. 9.7.

In this example, the Virtual Entities take the form of database records (Fig. 9.8) stored in a Network Resource, the EPC Database. This database is exposed for querying and updating through the EPCIS service (EPC Information Service).

The logistics manager, a Human User, can use the SCM application in order to view the status of the tracked items (pallets and cases). The SCM application is invoking the EPCIS query interface in order to get the necessary data.

Both pallet and cases have RFID tags attached that identify them. A RFID reader – a type of sensor – reads the EPCs on the tags and hosts a resource that makes the RFID inventory data accessible. A special service, the EPC Capturing Service, is exposing this resource and is updating the EPC Data Base by invoking the EPCIS capture interface of the EPCIS service. The EPCIS capture interface and the EPCIS query interface are standardized and defined by EPCglobal (EPC 1.0.13).

In principle other technologies for identification, e.g. visual ones like bar codes could be used. In this case, there is no hardware Device of type Tag involved and the Sensor would be a camera or barcode reader. The *identifies* relation (as in the IoT Domain Model) would then be directly between the Sensor and the Physical Entity. The other aspects would be modelled in the same way.

Finally note that also physical interactions with the pallet can take place: a warehouse worker – a Human User – moves around the pallet.

9.1.6 Examples for IoT Domain Model Concepts

In this section we give examples on different concepts in the IoT Domain Model. For each concept we discuss a practical example and, where applicable, we highlight the dependency of the concept on other concepts and also provide some general information.

9.1.6.1 User

A User interacts with a Physical Entity, physically or mediated through the IoT system. In the case of a mediated interaction, a User invokes or subscribes to a Service.

Application

- **Example:** A WSN installed in a wine cellar monitors environmental factors such as temperature, humidity, and light intensity. These factors play an essential role in defining the quality of the final wine product. Therefore, the winegrower has an intelligent application running on his smart phone. The application allows him to periodically visualize the status of the cellar. In this example, the application is a user and the cellar is a Physical Entity.
- **Note:** An application is one kind of Active Digital Artefact.

Human User

- **Example:** The employee in a supermarket loads the fridge with meat instead of cheese. Therefore, he regulates the temperature of the fridge accordingly. In this example, the employee is a Human User and the fridge is a Physical Entity.
- **Note:** The case of multiple Human Users for one Physical Entity is possible as well. We take the example of the safe in a bank. For security reasons, more than one high-ranked employee is required to identify themselves simultaneously at the safe in order to be able to open it. In this example the eligible employees are Human Users and the safe is the Physical Entity.

9.1.6.2 Physical Entity

A Physical Entity is a discrete, identifiable part of the physical environment which is of interest to the User for the completion of his goal. In the following different kinds of Physical Entities are discussed.

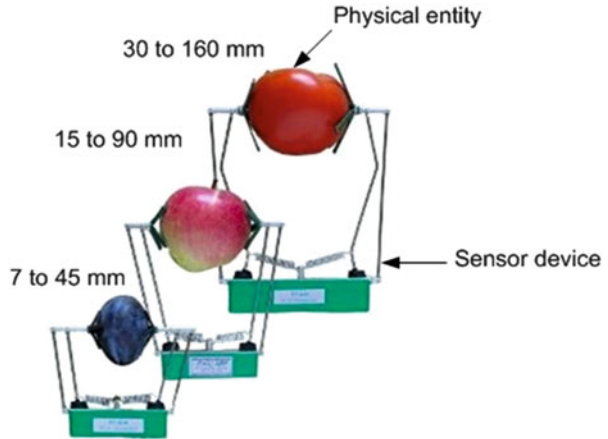
Environment

- **Example:** An optical fog sensor measures the density of water particles in the air that limit visibility. This sensor is used for traffic-control purposes, where it is often installed on the side of roads for monitoring visibility impairment through fog. The information about the fog is sent to a traffic management system where it is analyzed. In this example the near surrounding above the road is the Physical Entity.

Living Being

- **Example:** A WSN for agricultural monitoring. The network targets to report on the growth of fruits. To this end growth monitors are deployed. They are equipped with fruit-growth sensors as depicted in Fig. 9.9. In this example, the fruits are Physical Entities that are living beings.

Fig. 9.9 Growth fruit sensor © 2010 Phyto-Sensor Group



Structural Asset

- **Example:** Equipping bridges with electrochemical fatigue sensors that reveal flaws in metal (Phares 2007). This works much the same way as an electrocardiogram tests the human heart. First, bridge inspectors identify parts of the bridge that are more susceptible to cracks. Second, they equip these areas with electrochemical fatigue sensors. Third, they apply a constant electrical current that runs between the sensors and the bridge. By monitoring the amplitude of the current passing through the metal, sensors can detect cracks. In this example, a susceptible area of the bridge is a structural-asset Physical Entity.

9.1.6.3 Resource

Resources are software components that provide information about or enable the actuation on Physical Entities. We explain two examples for Resources, one illustrating an On-Device Resource and the other a Network Resource.

On-Device Resource

- **Example:** TinyOS is an event-based OS for embedded networked sensors (Levis and Gay 2009). TinyOS provides predefined software components that manage the access and control of i.e., local LEDs, radio, or sensors. In this example, the software components are On-Device Resources.

Network Resource

- **Example:** HBase² is an open-source, distributed, column-oriented database. HBase offers a set of functionalities that allow the management of distributed information. In this example the HBase software libraries and components are -Network Resources.

9.1.6.4 Service

A Service provides a well-defined and standardised interface, offering all necessary functionalities for interacting with Physical Entities and related processes. Often it exposes a functionality provided by a Resource to the overall IoT system.

Interacting Services

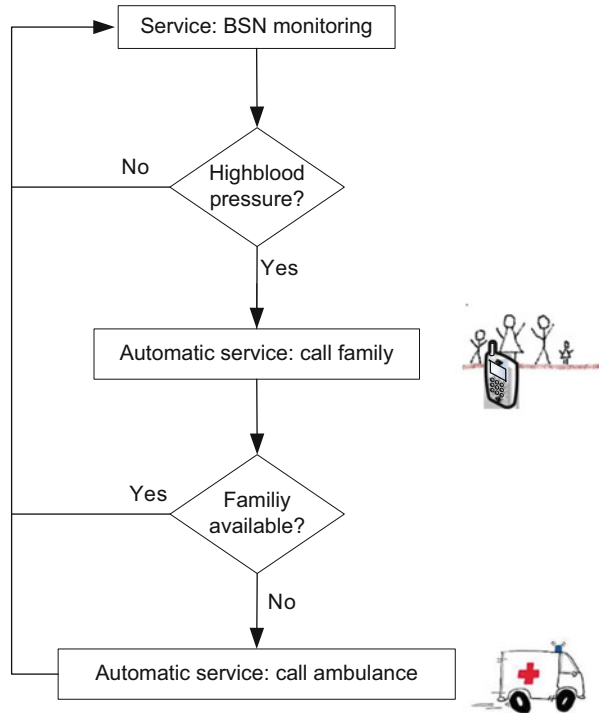
- **Example:** A system for home-patient monitoring. The system is composed of a body sensor network (BSN) attached to the body of the patient. Bioelectric chips monitor the status of the patient and require no direct involvement from a human being. As depicted in Fig. 9.10, the intelligence of the system resides not only in the hardware but also in three main services. First, the BSN monitoring service that evaluates the readings of the bioelectric chips i.e., a blood pressure. Second, the automatic service call, which alerts the relatives of the patient whenever his situation deteriorates. Third, another automatic service call that alerts the ambulance. The diagram in Fig. 9.10 shows the conditions to be fulfilled for one service to invoke another service.
- **Note:** A service demanding high processing and storage capabilities can be divided into multiple subservices running on different machines that invoke each other. In comparison to the original service, each of these subservices requires less storage and processing capabilities. Therefore, a trade-off exists between the number of subservices and the power consumption of the hosting machines. Distributed subservices induce an inter-communication overhead that increases the power-consumption of the hosting machines. This trade-off should be taken into consideration when dealing with low-power communicating devices (Polastre et al. 2005).

Service Associated with a Virtual Entity

- **Example:** Services can be associated with Virtual Entities and these associations are stored and can be discovered in the IoT system. The management of these associations can be handled in a centralized database or in a highly distributed fashion as in a peer-to-peer system, depending on the characteristics of the underlying system.

² <http://hbase.apache.org/>

Fig. 9.10 Interacting services for a home-patient monitoring scenario



Service Accessing a Resource

- **Example:** A service for monitoring air pollution. Sensor nodes are semi-randomly distributed in a city and measure the percentage of CO in the air. A remote server runs software that periodically queries the readings from the sensor nodes, analyses the readings, and monitors the evolution of the air pollution. In this example, the monitoring software is a service that accesses multiple resources. The latter are the components and functions running on sensor nodes, and these components allow operations such as reading from the sensors.

9.1.6.5 Device

Devices are technical artefacts, i.e. hardware, for bridging the real world of Physical Entities with the digital world of the Internet. Often a Device hosts Resources, which represent the software counterpart.

Devices

- **Example:** Typical devices are sensors, like temperature, noise or light sensors, but also more complex ones like cameras – or actuators, like switches, door openers or more complex ones like air conditioning systems.

Hierarchical Devices

- **Example:** As depicted in Fig. 9.11, a Telos node contains three types of integrated sensors (photodiode, humidity and temperature), several expansion pins to mount external sensors, and three integrated LEDs (Polastre 2005). Two views of the node exist: The node as a whole may be seen as a single device or it can be seen as a composition of multiple sensors and actuators acting as individual devices.
- **Note:** A device can be seen as a single unit as well as a composition of multiple devices. This granularity of modulating a device is not specified in the IoT Domain Model due to the fact that it is application dependent.

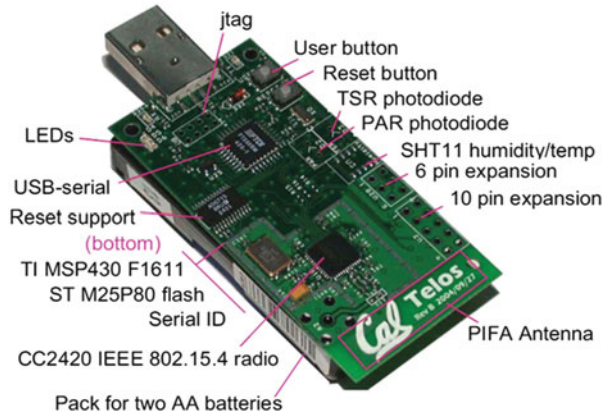
9.1.6.6 Deployment Configurations

Figure 9.12 shows a range of deployment configurations for resources, services, and users. In Fig. 9.12 (a) resource, service, and the user (application) are running on the same device. This is a configuration in which we have a powerful device, and the interaction with the user is local. In Fig. 9.12 (b) the service of the user is running somewhere else, e.g., in the cloud, and the interaction is thus not local. The API used between the service client and the service, however, is the same. In Fig. 9.12 (c) the service is not running on the device, but in the cloud. This is a typical configuration for a constrained device that may not be able to expose a user interface across the network. For example, due to energy constraints or other limiting factors, such a device may sleep most of the time and is therefore not able to always handle user requests. The interface between the service and the resource may be very specific and proprietary.

Network-based resources are not shown in Fig. 9.12, as they can be regarded as being hidden behind cloud-based services.

Of course, in a real IoT system all these different configurations may be realized at the same time and there could be interactions between users and services from the different configurations.

Fig. 9.11 Telos ultra-low power wireless module © 2008 University of California, Berkeley



9.1.7 Generating a Specific IoT Domain Model

As discussed in Sect. 6.3, the IoT Domain Model is an integral part of the IoT architecting process. In the following we provide a six-step process that supports the generation of use-case specific IoT Domain Models. In the following, we illustrate the answers with examples from the recurring example that we introduced in Sect. 4.2.

In order to proceed with the modelling of a system, its usage from the perspective of each User needs to be analysed. For each of the Users identified, the architect needs to answer six simple questions, and create suitable instance diagrams from the Domain Model based on the answers.

Q1: What does a User invoke or subscribe to?

A1: The answer determines the Service(s) that the user invokes or subscribes to – In the recurring example the user subscribes to the alarm service (using an Android app).

Q2: Which part of the environment does the User want to interact with?

A2: The answer determines the PE(s) – in the recurring example the user wants to be kept informed about the status of the load carrier.

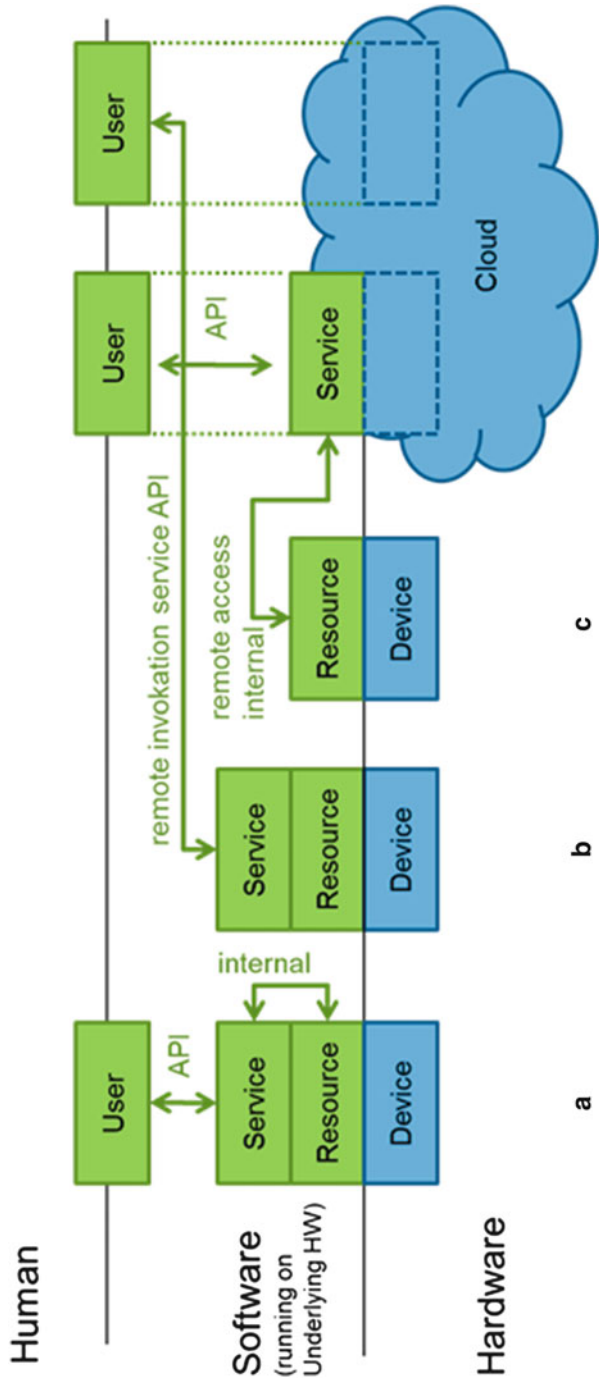
Q3: What is used to identify/monitor this PE in the physical world?

A3: The answer determines the Device(s) – in the recurring example, the load carrier can be identified with an RFID, a humidity sensor and a temperature sensor can monitor relevant state information.

Q4: What is used to identify the PE in the digital world?

A4: The answer determines the VE(s) – in the digital world the identifier provided by the RFID can be used for the VE modelling the load carrier.

Fig. 9.12 Various deployment configurations of devices, resources, and services



Q5: What software can provide information or allow changing aspects related to PE?

A5: The answer determines the Resource(s) – the Alarm Resource can trigger a notification if the temperature or the humidity are no longer within the required range.

Q6: What exposes this Resource and/or makes it accessible?

A6: The answer determines Resource-level Service(s) – the Alarm Service exposes the Alarm Resource.

9.2 Usage of the IoT Information Model

The IoT Information Model cannot be instantiated directly like the IoT Domain Model. Moreover the IoT Information Model defines an abstract framework or meta-model that is technology agnostic and restricted to a minimum. The model is just enough to accommodate the relationships defined in the IoT Domain Model and to model the key concepts that are used as a basis for defining interfaces of functional components. Thus only the skeleton of an information model is provided in the ARM that IoT-A compliant architectures will have in common. A common model on the other hand can serve as a bridge between more specific -but different -information models to be used in concrete architectures.

The way to work with the IoT Information Model is split into three steps (see also Fig. 9.13 below):

1. Use the IoT Information Model, viz. meta-model, as a basis explaining the common information structure and the core elements defined in the IoT Domain Model, like Virtual Entities, Attributes and Services;
2. Generate a domain-specific information model out of the meta-model, which defines a minimal set of attributes and Services for your application domain. Attributes which every VE needs to have (e.g. `EntityId` or `EntityType` as in Chap. 7 Fig. 9.10 “IoT Information Model”) are defined but not necessarily described in detail. Additionally the Service Descriptions can already be defined as interfaces with input and output parameters.
3. Several (different) representations of the domain-specific model can be generated, implementing the defined Attributes and Services. The use of different representations is useful when there are different implementation-specific requirements, like binary storage of information for constrained sensor nodes and XML storage for the backend server storage.

The IoT Information View in Sect. 8.2.3, and especially the Sect. 8.2.3.1, give some examples how the concrete modelling of the domain-specific model can look like. An additional example for an information model used to model events is shown in (Voelksen 2013; Sect. 4.2).

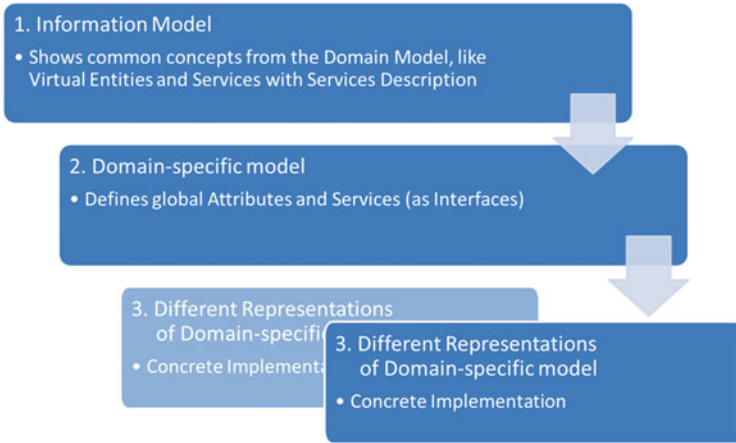


Fig. 9.13 Three steps to use the IoT information model

9.3 Usage of the IoT Communication Model

Extending the analysis performed using the other models the IoT Communication Model is used in the architectural design process to:

1. Identify homogeneous sub-systems and their capabilities and constraints;
2. Identify suitable protocol stacks and network topologies to be merged in a common system view;
3. Define gateways and other bridging solutions.

9.3.1 Guidelines for Using the IoT Communication Model

Since the IoT Communication Model aims at providing an overall framework for communication in IoT systems, it requires well-defined domain and information definitions. This can be achieved following the examples of the previous sections. Starting from those, it is possible to identify all the sub-systems, the complete system is composed of, where we define homogeneous sub-system as a set of system elements sharing the same communication technology and sharing similar hardware capability.

Once the sub-systems have been defined, it is possible to analyse capabilities and constraints for each of them. By capabilities and constraints we intend communication specific parameters such as data rate, delays, medium reliability (channel errors) and technology specific parameters such as the available memory, computational power and supported functionalities.

Modelling Rule 4	Identify homogeneous sub-systems from the complete domain model and determine their capabilities and constraints.
---------------------	---

Subsequently, it is possible to analyse communication requirements deriving from both services in the domain definition and interaction patterns from the information model. The main goal of the IoT Communication Model is to identify a set of interoperable protocol stacks and topologies with the following characteristics:

1. Each stack must grow from a specific communication technology;
2. Interoperability shall be enforced in the lowest possible layer of stack;
3. The combination of identified stacks and topologies must satisfy all the requirements.

Modelling Rule 5	Use existing standard communication mechanisms and related protocols whenever possible. If this is not possible then each of the sub-system is the starting point for building a protocol stack which is both technology specific and interoperability prone.
---------------------	---

This rule enforces technology optimizations and ensures feasibility in all the subsystems.

Modelling Rule 6	Interoperability shall be enforced in the lowest possible layer.
---------------------	--

This rule enforces simplicity and interoperability, because it avoids stack duplication and makes it possible to reuse the same protocols (and their implementations) horizontally in the system. Usually, the most effective interoperability point is the Network & ID aspect in the IoT Communication Model (or the Network layer in the ISO/OSI model) as it is the lowest common point in the stack which is not technology specific and, thus, it can be the same across different sub-systems.

For such a reason the selection of the protocols governing the Network & ID aspect is of paramount importance, since they must satisfy the requirements from services and respect the technology constraints.

The next aspect in the IoT Communication Model is the end-to-end aspect: this considers every possible interaction path among any couple of sub-systems. Again, technology dependent constraints and service dependent requirements will push the system architect in two opposite directions and often there is no single rule for all

the systems: in fact, even though the Network & ID aspect is capable of making two sub-systems communicate to one another, it is the end to end aspect which harmonizes the overall system behaviour. For such a reason, this is the place where gateways and proxies are mostly needed.

Modelling Rule 7	In order to allow seamless interaction between sub-systems, gateway and proxies shall be designed for the whole system.
------------------	---

Finally, the Data interoperability aspect of the IoT Communication Model, which accounts for the highest layer of the ISO/OSI communication stack, considers the remaining aspects of data exchange, compression and representation. Although application layer gateways can always be designed to map two different data representations, it is not always advisable to do so. Most often adopting a compressed format which fits constrained network capabilities provides two advantages, (1) simpler network interactions, and (2) lower traffic.

9.4 Usage of Perspectives

Perspectives are used to help an architect in designing software architectures. Their purpose is threefold (Rozanski and Woods 2011):

1. Standardised store of knowledge about a given quality property;
2. As a guide for architects not experienced with a given quality property, and
3. As a memory aid for the experienced architecture.

The actual use of perspectives in an architectural design process is shown in Fig. 9.14. Within the IoT-A project we extended the use of perspectives by adding another layer: the Design Choices catalogue. Design Choices which are very concrete usages of the IoT Reference Architecture applied to Functionality Groups and Functional Components. An architect can consider solutions provided by the Design Choices when creating the initial candidate architecture and later on when he is modifying the architecture to resolve the problems with unacceptable quality properties.

The architect designing should always keep the desired use of the system into account. For example, the architect designing the system used in the “Red Thread” example from Sect. 4.2 would go through the scenarios with a specific “hat” for all perspectives. He would first extract the non-functional requirements (e. g. the reliability needs of the sensors, security concerns) and then, once he has a candidate architecture, use the perspectives to ensure that on all the non-functional requirements have been taken care of. He would, for example, have a specific look at the safe storage of the sensor history and select a Design Choice which ensures that it cannot be altered. The perspectives then will help him making sure

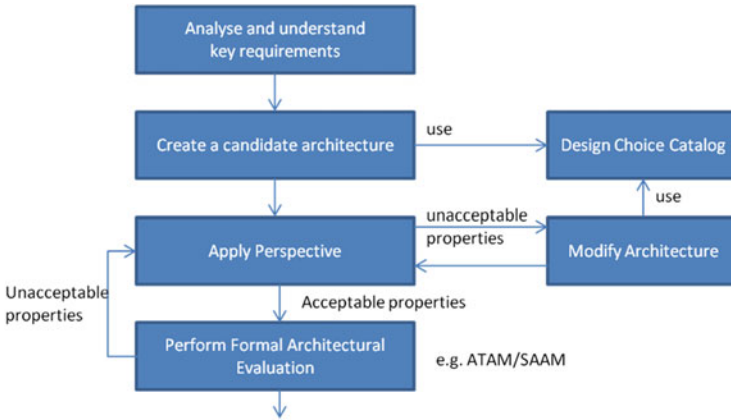


Fig. 9.14 Using perspectives (Adopted from (Rozanski 2011))

that still all other requirements are fulfilled, and if not, at least can help making the trade-offs explicit.

Applying a perspective is more than a review process: the outcome of applying a perspective is cross-view changes to the architecture. As an additional outcome of the perspectives there might be additional documents like performance analysis data etc.

Open Access This chapter is distributed under the terms of the Creative Commons Attribution Noncommercial License, which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.