Chapter 7
IoT Reference Model

Martin Bauer, Nicola Bui, Jourik De Loof, Carsten Magerkurth, Andreas Nettsträter, Julinda Stefa, and Joachim W. Walewski

7.1 Introduction

The first major contribution of the *IoT Architectural Reference Model* (IoT ARM) is the IoT Reference Model itself. Besides models, the IoT Reference Model provides the concepts and definitions on which IoT architectures can be built. This

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

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

J. De Loof
Alcatel-Lucent Bell N.V., Copernicuslaan 50, Antwerpen 2018, Belgium
e-mail: jourik.de_loof@alcatel-lucent.com

C. Magerkurth
SAP AG, Dietmar-Hopp-Allee 16, Walldorf 69190, Germany
e-mail: carsten.magerkurth@sap.com

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

J. Stefa
Università Sapienza di Roma, P.le Aldo Moro 5, Rome 00185, Italy
e-mail: stefa@di.uniroma1.it

J.W. Walewski
Siemens AG, Otto-Hahn-Ring 6, Munich 81739, Germany
e-mail: joachim.walewski@siemens.com; www.siemens.com

Chapter introduces the IoT Reference Model as a precondition for working with the Reference Architecture that is introduced in Chap. 8.

The Reference Model consists of several sub-models that set the scope for the IoT design space and that address architectural views and perspectives discussed in Chap. 8. As already stated above, the primary and thus the key model is the IoT Domain Model, which describes all the concepts that are relevant in the Internet of Things. All other models and the IoT Reference Architecture are based on the concepts introduced in the IoT Domain Model. While certain models, such as the IoT Communication Model and the IoT Trust, Security, and Privacy Model might be less critical in certain application scenarios, the IoT Domain Model is mandatory for all usages of the IoT ARM. Therefore, it is advised to read Sect. 7.1.3 carefully, and at least to follow the information given in the Sect. 7.1.2 in order to get an overview of the different sub-models of the IoT Domain Model and how they relate to each other. Depending on the individual application of the IoT Domain Model, the Subsequent sections in this chapter provides details about the other models.

Next, we explain, who the sub-models in the IoT Reference Model relate and link to each other, and how they form an integrated reference model.

7.2 Interaction of All Sub-Models

The IoT Reference Model aims at establishing a common grounding and a common language for IoT architectures and IoT systems. It consists of the sub-models shown in Fig. 7.1, which we explain below. The yellow arrows show how concepts and aspects of one model are used as the basis for another.

The foundation of the IoT Reference Model is the IoT Domain Model, which introduces the main concepts of the Internet of Things like Devices, IoT Services and *Virtual Entities* (VE), and it also introduces relations between these concepts. The abstraction level of the IoT Domain Model has been chosen in such a way that its concepts are independent of specific technologies and use-cases. The idea is that these concepts are not expected to change much over the next decades or longer.

Based on the IoT Domain Model, the IoT Information Model has been developed. It defines the structure (e.g. relations, attributes) of IoT related information in an IoT system on a conceptual level without discussing how it would be represented. The information pertaining to those concepts of the IoT Domain Model is modelled, which is explicitly gathered, stored and processed in an IoT system, e.g. information about Devices, IoT Services and Virtual Entities.

The IoT Functional Model identifies groups of functionalities, of which most are grounded in key concepts of the IoT Domain Model. A number of these *Functionality Groups* (FG) build on each other, following the relations identified in the IoT Domain Model. The Functionality Groups provide the functionalities for interacting with the instances of these concepts or managing the information related to the concepts, e.g. information about Virtual Entities or descriptions of IoT Services. The functionalities of the FGs that manage information use the IoT Information Model as the basis for structuring their information.

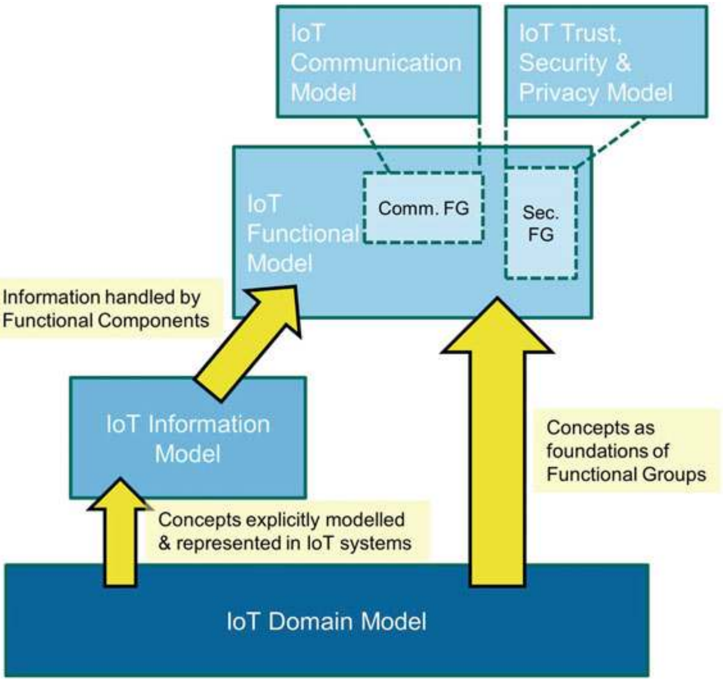

IoT Communication Model
IoT Trust, Security & Privacy Model
IoT Functional Model
Comm. FG
Sec. FG
Information handled by Functional Components
IoT Information Model
Concepts as foundations of Functional Groups
Concepts explicitly modelled & represented in IoT systems
IoT Domain Model

Fig. 7.1 Interaction of all sub-models in the IoT Reference Model. The sub-models are explained in the text body

A key functionality in any distributed computer system is the communication between the different components. One of the characteristics of IoT systems is often the heterogeneity of communication technologies employed, which often is a direct reflection of the complex needs such systems have to meet. The IoT Communication Model introduces concepts for handling the complexity of communication in heterogeneous IoT environments. Communication also constitutes one FG in the IoT Functional Model.

Finally, *Trust*, *Security and Privacy* (TSP) are important in typical IoT use-case scenarios. Therefore, the relevant functionalities and their interdependencies and interactions are introduced in the IoT TSP Model. As in the case of communication, security constitutes one FG in the Functional Model.

7.3 Domain Model

7.3.1 Definition and Purpose

The IoT-A project defines a domain model as a description of concepts belonging to a particular area of interest. The domain model also defines basic attributes of these concepts, such as name and identifier. Furthermore, the domain model defines relationships between concepts, for instance "*Services* expose *Resources*". Domain models also help to facilitate the exchange of data between domains (The Consultative Committee 2006). Besides this official definition, and looking at our interpretation of it, our domain model also provides a common lexicon and taxonomy of the IoT domain (Muller 2008). The terminology definitions of IoT-A are provided online (Sect. 6.7).

The main purpose of a domain model is to generate a common understanding of the target domain in question. Such a common understanding is important, not just project-internally, but also for the scientific discourse. Only with a common understanding of the main concepts it becomes possible to argue about architectural solutions and to evaluate them. As has been pointed out in literature, the IoT domain suffers already from an inconsistent usage and understanding of the meaning of many central terms (Haller 2010).

The domain model is an important part of any reference model since it includes a definition of the main abstract concepts (abstractions), their responsibilities, and their relationships. Regarding the level of detail, the Domain Model should separate out what does not vary much from what does. For example, in the IoT domain, the device concept will likely remain relevant in the future, even if the types of devices used will change over time and/or vary depending on the application context. For instance, there are many technologies to identify objects: RFID, bar codes, image recognition etc. But which of these will still be in use 20 years from now? And which is the best-suited technology for a particular application? Since no one has the answers to such and related questions, the IoT Domain Model does not include particular technologies, but rather abstractions thereof.

Before we discuss the main abstractions and relationships of the IoT Domain Model in detail, let us go back to our recurring example that we introduced in Sect. 4.2 in order to get an understanding of what it means to formulate central concepts of a use case with the help of the IoT Domain Model.

Figure 7.2 shows an instance diagram of central aspects of the use case scene in Sect. 4.2. This example was cast in the language of the IoT Domain Model and then illustrated by use of UML. Information about UML can be found elsewhere in the literature (Fowler 2003) or by searching for terms such as "UML tutorial" on the web.

As we can see in Fig. 7.2, the important entities that are relevant for our use case are depicted with blocks of different colours. For instance, there is our truck driver "Ted" represented by as a yellow box (viz. instance), and the temperature sensor (that triggers an alarm after Ted had turned off the engine of the truck) is represented as a blue instance. Already at this stage we can easily deduct that there is some colour-coding involved that reflects an aspect of the respective entity. What these colours exactly stand for is discussed in detail in the next Sections. There is also a categorisation in textual form, as the entity name that we know from our recurring example is succeeded by an entity category such as Sensor in the case of the humidity or temperature sensors and *Human User* in the case of Ted. What these entity categories mean and how they relate to each other is discussed in detail in the next sections.

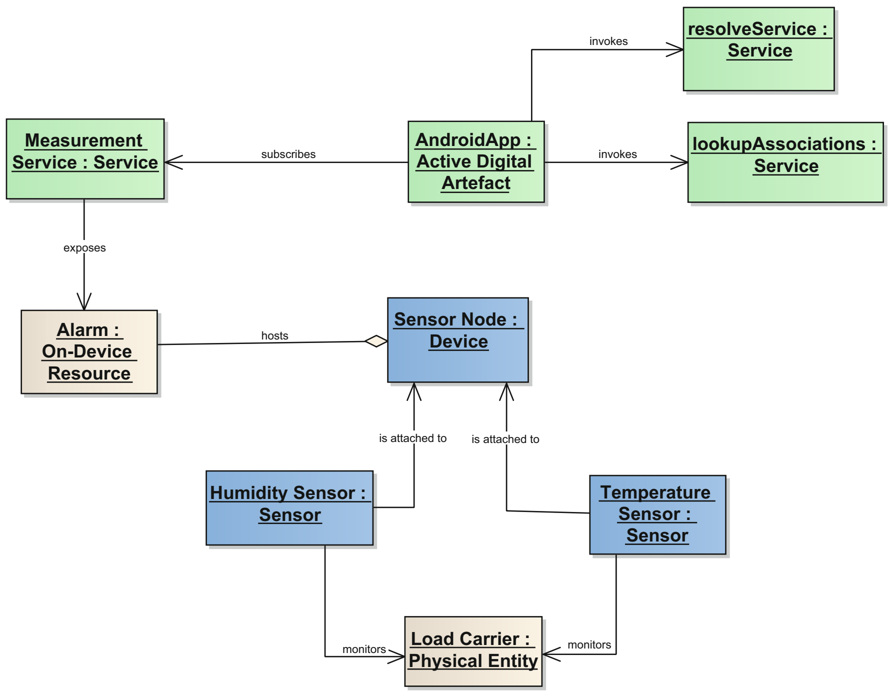

resolveService : Service
invokes
Measurement Service : Service
subscribes
AndroidApp : Active Digital Artefact
invokes
lookupAssociations : Service
exposes
Alarm : On-Device Resource
hosts
Sensor Node : Device
is attached to
is attached to
Humidity Sensor : Sensor
Temperature Sensor : Sensor
monitors
Load Carrier : Physical Entity
monitors

Fig. 7.2 Example, instantiated IoT Domain Model for the Red Thread Example (see Sect. 4.2)

In addition to the coloured boxes, the diagram also shows arrows with verbs that connect the boxes. If we look very closely to the arrows, we see that they have different terminators such as diamond shapes or traditional arrow shapes. These shapes illustrate different kinds of relationships between the objects that are connected by them. In a similar way as the category names and the colour coding of the objects are related to each other, the verbs indicate information about the relationships shown with the arrows. These are all concepts of the UML notation that will be discussed in the next section.

Even without understanding all of the concepts in detail, we can already understand that the IoT Domain Model helps us structuring an application scenario. We can use a concise graphical representation to show that for instance Ted, our truck driver, is a Human User that uses an Android application in order to subscribe to an Alarm service. This Android Application is an *Active Digital Artefact* (*ADA*). We do not yet know what this exactly means, but as the reader will progresses through this document and possibly other documents that make use of the IoT Domain Model, Active Digital Artefacts will come up again and again. By providing a standardised vocabulary for naming things that relate to the same abstract concepts, we facilitate and streamline communication of the IoT ARM users.

While several other parts of the IoT Reference Model, for instance the IoT Information Model, directly depend on the IoT Domain Model, and also several views (as we will see in the next chapter), it should already be noted that the IoT Domain Model also takes a central role in the process of generating concrete architectures beyond merely providing a common language. As discussed in Chap. 6, Sect. 6.3, there is a special view called IoT Context View that is central in the process of generating concrete architectures. This view is an amalgam of the IoT Domain Model "traditional" context view. The latter is an architecture view that is usually generated at the very beginning of the architecture process. It describes "the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts)" (Rozanski and Woods 2011).

7.3.2 Main Abstractions and Relationships

7.3.2.1 Interpreting the Model Diagram

This section describes the IoT Domain Model used in the IoT-A project. It was developed by refining and extending two models found in the literature (Haller 2010; Serbanati et al. 2011). The goal behind the IoT Domain Model is to capture the main concepts and the relationships that are relevant for IoT stakeholders. After a short introduction to the pertinent UML language (next Section), we expatiate the IoT terminology and concepts in Sect. 7.1.3.3. A discussion about guidelines and best practices on how to use the IoT Domain Model are provided in Chap. 9.

UML is used to graphically illustrate the model (Fowler 2003). Generalisation is used to depict an is-a relationship and should not be misinterpreted as sub-classing. Only the most important specialisations are shown, others are possible however. For example, not every *Device* can be characterised as a Tag, a Sensor, or an Actuator. The specialisations are, however, generally disjoint, if not noted otherwise.

Please note that generalisations/specialisations are modelled using a solid line with a large hollow triangle.

The notation indicates that class A is the Parent or super-class, while class B and class C are child or subclasses. Objects represented by class B and class C "inherit" the attributes of the object represented by class A (their parent), while having additional unique attributes of their own. This relationship is referred to as the is-a relationship – an object in class B or class C is-a type of class A (see Fig. 7.3).

This notation is not to be confused with an "aggregation or composition relationships". Rather, a terminating "open diamond" indicates an aggregation relationship, whereas a "filled diamond" indicates a composition relationship. The notation in Fig. 7.4 states that class A is an aggregation of (or contains) objects of class B and a composition of objects of class C. In other words, class A has-a class B and also class C is-part of class A. Aggregation and composition are rather similar, however the lifetime of objects of class C is determined by class A

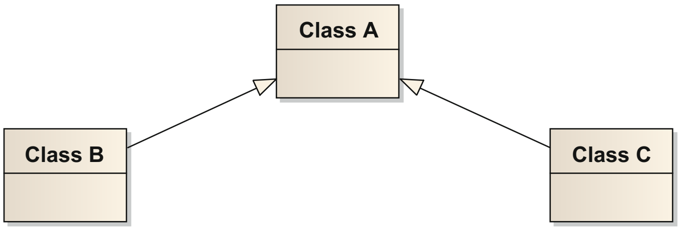


Fig. 7.3 UML generalization

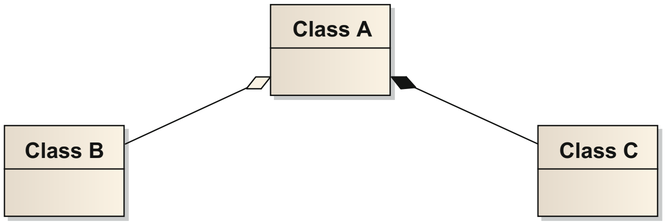


Fig. 7.4 UML aggregation and composition

("A brain is part of a student" -> composition), whereas the lifetime of objects of class B is independent from class A ("A student has a school").

Finally, an "open arrow" is used to denote a "one-way" association. The notation shown in Fig. 7.5 indicates that every object in class A is associated with zero or more objects in class B, and that every object in class B is associated with exactly one object in class A. However more importantly, this notation indicates that a class A object will "know" class B objects with which it is associated, and that a class B object will "not know" the class A object with which it is associated, ref. Sensor and Physical Entity in Fig. 7.7.

The cardinalities ("asterisk", "1", etc.) are to be read as follows: from the source read the relation and the cardinality on the target gives the multiplicity with which the source can be in that relation with the target. For the inverse relation, the cardinality at the source is relevant. For example (see Fig. 7.7), a Tag identifies no or one (0..1) Physical Entity – whereas *a* Physical Entity may be identified by 0 or more Tags. A Virtual Entity may contain 0 or more other Virtual Entities, whereas a Virtual Entity can optionally be contained in at most one other Virtual Entity. Concepts depicting hardware are shown in blue, software in green, animate beings in yellow, and concepts that fit into either multiple or no categories in brown.

7.3.2.2 The Concepts of the IoT Domain Model

The most generic IoT scenario can be identified as that of a generic *User* needing to interact with a (possibly remote) *Physical Entity* (PE) in the physical world

Fig. 7.5 UML association

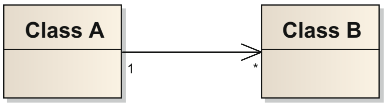
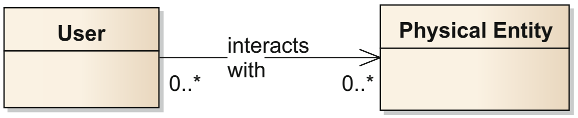


Fig. 7.6 Basic abstraction of an IoT interaction



(see Fig. 7.6). In this short description we have already introduced the two key entities of the IoT. The User is a human person or some kind of a Digital Artefact (e.g., a *Service*, an application, or a software agent) that needs to interact with a Physical Entity.

In the physical environment, interactions can happen directly (e.g., by moving a pallet from location X to Y manually). In the IoT though, we want to be able to interact indirectly or *mediated*, i.e., by calling a Service that will either provide information about the Physical Entity or act on it. When a Human User is accessing a service, he does so through a service client, i.e., software with an accessible user interface. For the sake of clarity, the service client is not shown in Fig. 7.7. For the scope of the IoT Domain Model, the interaction is usually characterised by a goal that the User pursues. The Physical Entity is an identifiable part of the physical environment that is of interest to the User for the completion of her goal. Physical Entities can be almost any object or environment; from humans or animals to cars; from store or logistics chain items to computers; from electronic appliances to jewellery or clothes.

Physical Entities are represented in the digital world by a Virtual Entity. This term is also referred to as "virtual counterpart" in the literature (Römer et al. 2002), but using the same root term "entity" in both concepts clearer shows the relationship of these concepts. There are many kinds of digital representations of Physical Entities: 3D models, avatars, database entries, objects (or instances of a class in an object-oriented programming language), and even a social-network account could be viewed as such a representation, because it digitally represents certain aspects of its human owner, such as a photograph or a list of his hobbies. However, in the IoT context, Virtual Entities have two fundamental properties:

- They are Digital Artefacts. Virtual Entities are associated to a single Physical Entity and the Virtual Entity represents this very Physical Entity. While there is generally only one Physical Entity for each Virtual Entity, it is possible that the same Physical Entity can be associated to several Virtual Entities, e.g., a different representation per application domain. Each Virtual Entity must have one and only one ID that identifies it univocally. Virtual Entities are Digital Artefacts that can be classified as either active or passive. *Active Digital Artefacts* (ADA) are running software applications, agents or Services that may access other Services or Resources. *Passive Digital Artefacts* (PDA) are passive software elements such as database entries that can be digital

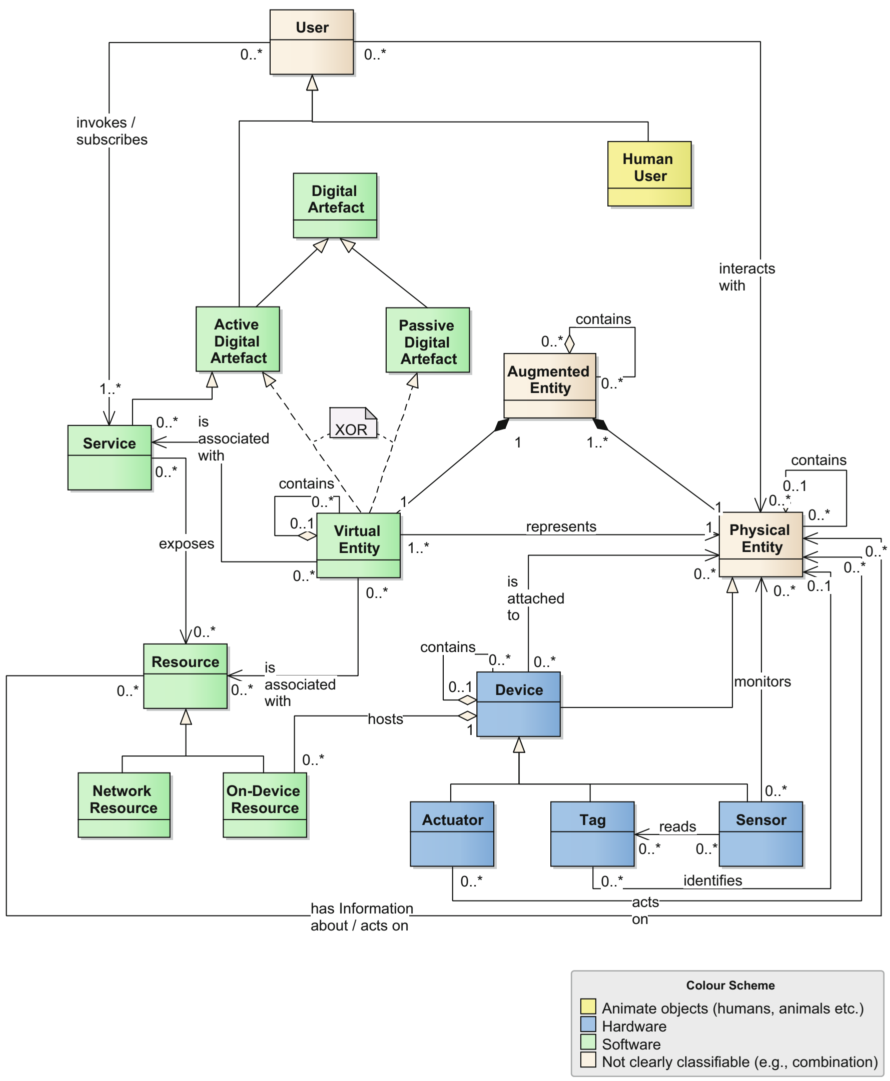


Fig. 7.7 UML representation of the IoT Domain Model

representations of the Physical Entity. Please note that all Digital Artefacts can be classified as either Active or Passive Digital Artefacts;

- Ideally, Virtual Entities are synchronised representations of a given set of aspects (or properties) of the Physical Entity. This means that relevant digital parameters representing the characteristics of the Physical Entity are updated upon any change of the former. In the same way, changes that affect the Virtual Entity could manifest themselves in the Physical Entity. For instance, manually locking a door might result in changing the state of the door in home automation software, and correspondingly, setting the door to "locked" in the software might result in triggering an electric lock in the physical world.

At this point it should be noted that while Fig. 7.6, at first sight, seems to suggest only a Human User interacting with some Physical Entities, it also covers interaction between two machines: in this case, the controlling software of the first machine is an Active Digital Artefact and thus a User, and the second machine – or a Device in the terms of the IoT Domain Model – can be modelled as a Physical Entity. We introduce the concept of an Augmented Entity as the composition of one Virtual Entity and the Physical Entity it is associated to, in order to highlight the fact that these two concepts belong together. The Augmented Entity is what actually enables everyday objects to become part of digital processes, thus, the Augmented Entity can be regarded as constituting the "thing" in the Internet of Things.

It should be noted that there might be many types of users, as we have discussed before. A Human User is a specialisation of the general concept. However, different kinds of Users, such as maintenance people, owners, or security officers are plausible as well. It is also worth noting that we have not included different roles in the IoT Domain Model, for same reason that we have also not introduced different types of Users. Within the development of concrete architectures, it is very likely that the Users will take on different roles and these should be modelled accordingly. As the underlying taxonomies will vary with the use cases addressed, we do not prescribe a specific taxonomy here. Especially in the enterprise domain, where security roles are fundamental to practically every single IoT architecture, one common option for modelling roles can be found in (Raymond 1995). We will briefly revisit up this taxonomy within the context of the process management Section (see Sect. 7.1.5.2.1).

The relationship between Augmented, Physical and Virtual Entities is shown in Fig. 7.7, together with other terms and concepts that are introduced in the remainder of this section.

The relation between Virtual Entity and Physical Entity is usually achieved by embedding into, by attaching to, or by simply placing in close vicinity of the Physical Entity, one or more ICT Devices that provide the technological interface for interacting with, or gaining information about the Physical Entity. By so doing the Device actually extends the Physical Entity and allows the latter to be part of the digital world. This can be achieved by using Devices of the same class, as in the case of certain similar kinds of body-area network nodes, or by using Devices of different classes, as in the case of an RFID tag and reader. A Device thus mediates the interactions between Physical Entities (that have no projections in the digital world) and Virtual Entities (which have no projections in the physical world), generating a paired couple that can be seen as an extension of either one, i.e. the Augmented Entity. Devices are thus technical artefacts for bridging the real world of Physical Entities with the digital world of the Internet. This is done by providing monitoring, sensing, actuation, computation, storage and processing capabilities. It is noteworthy that a Device can also be a Physical Entity, especially in the context of certain applications. An example for such an application is Device management, whose main concern is the Devices themselves and not the entities or environments that these Devices monitor.

From an IoT point of view, the following three basic types of Devices are of interest:

- **Sensors** provide information, knowledge, or data about the Physical Entity they monitor. In this context, this ranges from the identity of the Physical Entity to measures of the physical state of the Physical Entity. Like other Devices, they can be attached or otherwise embedded in the physical structure of the Physical Entity, or be placed in the environment and indirectly monitor Physical Entities. An example for the latter is a face-recognition enabled camera. Information from *sensors* can be recorded for later retrieval (e.g., in a storage of Resource);
- **Tags** are used to identify Physical Entities, to which *the* Tags are usually physically attached. The identification process is called "reading", and it is carried out by specific Sensor Devices, which are usually called readers. The primary purpose of Tags is to facilitate and increase the accuracy of the identification process. This process can be optical, as in the case of barcodes and QR codes, or it can be RF-based, as in the case of microwave car-plate recognition systems and RFID. The actual physics of the process, as well as the many types of tags, are however irrelevant for the IoT Domain Model as these technologies vary and change over time. These are important however when selecting the right technology for the implementation of a concrete system;
- **Actuators** can modify the physical state of a Physical Entity, like changing the state (translate, rotate, stir, inflate, switch on/off,...) of simple Physical Entities or activating/deactivating functionalities of more complex ones.

Notice though that Devices can be aggregations of several Devices of different types. For instance, what we call a sensor node often contains both Sensors (e.g., movement sensing) as well as Actuators (e.g., wheel engines). In some cases, Virtual Entities that are related to large Physical Entitie*s* might need to rely on several, possibly heterogeneous, Resources and Devices in order to provide a meaningful representation of the Physical Entity, c.f. in our Red Thread example, the values of several temperature Sensors are aggregated to determine the temperature of the truck.

Resource*s* are software components that provide data from or are used in the actuation on Physical Entities. Resources typically have native interfaces. There is a distinction between On-Device Resources and Network Resources. As the name suggests, On-Device Resources are hosted on Devices, viz. software that is deployed locally on the Device that is associated with the Physical Entity. They include executable code for accessing, processing, and storing Sensor information, as well as code for controlling Actuators. On the other hand, Network Resources are Resources available somewhere in the network, e.g., back-end or cloud-based databases. A Virtual Entity can also be associated with Resources that enable interaction with the Physical Entity that the Virtual Entity represents.

In contrast to heterogeneous Resources – implementations of which can be highly dependent on the underlying hardware of the Device – , a Service provides an open and standardised interface, offering all necessary functionalities for interacting with the Resources / Devices associated with Physical Entities. Interaction with the Service is done via the network. On the lowest level – the one interfacing with the Resource and closer to the actual Device hardware – , Services expose the functionality of a Device through its hosted Resources. Other Services may invoke such low-level Services for providing higher-level functionalities, for instance executing an activity of a business process. A typical case for this is the Service alerting "Ted" based on the temperature Service results in the "Red Thread" example.

Since it is the Service that makes a Resource accessible, the above-mentioned relations between Resources and Virtual Entities are modelled as associations between Virtual Entities and Services. For each Virtual Entity there can be associations with different Services that may provide different functionalities, like retrieving information or enabling the execution of actuation tasks. Services can also be redundant, i.e., the same type of Service may be provided by different instances (e.g. redundant temperature Services provided by different Devices). In this case, there could be multiple associations of the same kind for the same Virtual Entity. Associations are important in look-up and discovery processes.

The instance diagrams such as Fig. 7.2 are concrete instantiations of the IoT Domain Model, i.e. concrete architectures modelled with the concepts of the IoT Domain Model.

7.3.3 Detailed Explanations and Related Concepts

The IoT Domain Model as explained in the previous section is focusing on the main concepts at a high level of abstraction, capturing the essence of the IoT Domain. However, for easier understanding we provide here more detailed explanations.

7.3.3.1 Devices and Device Capabilities

From an IoT Domain-Model point of view, Devices are only technical artefacts meant to provide an interface between the digital and the physical worlds, i.e. a link between the Virtual Entities and the Physical Entities. For this reason, Devices must be able to operate both in the physical and digital world and the IoT Domain Model only focuses on their capability to provide observation and modification of the physical environment from the digital environment. If other properties of Devices were relevant, the Device would be modelled as an entity itself.

The hardware underlying the Devices is very important though and must have at least some degree of communication, computation and storage capabilities for the purposes of the IoT. Moreover, power resources are also very important, as they can provide operational autonomy to the Devices. Many technologies and products are available and their capabilities vary noticeably. While these capabilities might not impact directly the IoT Domain Model, they are very important during the application-design phase, c.f. the Deployment and Operation view in Sect. 8.2.4 “Deployment & Operation view”.

Communication capabilities depend on the type of data exchanged with the *Device* (identifier, identifier + data, sensor data, or commands) and the communication topology (network, reader-tag, peer-to-peer, etc.). These aspects are very important in the IoT context and have a large impact on energy consumption, data-collection frequency, and the amount of data transmitted. Communication capabilities indirectly impact the location of *Resources* (on-device or on the network). Please refer to the IoT Communication Model (Sect. 7.1.6) for a detailed discussion of this topic. Security features also impact communication capabilities, since they usually introduce a relevant communication overhead (c.f. Sect. 7.1).

Computation capabilities on the other hand have a huge impact on the chosen architecture, the implementable security features, and power resources of the Devices. They are also relevant for what concerns the availability of On-Device Resources and their complexity, as constrained Devices might not have sufficient computational resources.

The term storage usually refers to the capability of supporting the firmware/ software running on the Device. This can be accomplished storing data provided by on-board sensor hardware or data gathered from other Services and needed for supporting a given Resource. Storage can range from none, as in the case of RFID technology to kilobytes in the case of typical embedded Devices or even more in case of unconstrained Devices.

7.3.3.2 Resources

Resources are software components that provide some functionality. When associated with a Physical Entity, they either provide some information about or allow changing some aspects in the digital or physical world pertaining to one or more Physical Entities. The latter functionality is commonly referred to as actuation. Resources can either run on a Device – hence called On-Device Resources – or they can run somewhere in the network (Network Resources). On-Device Resources are typically sensor Resources that provide sensing data or actuator Resources, e.g. a machine controller that effects some actuation in the physical world. They thus can be seen as a “bridge” between the digital and physical world. On-Device Resources may also be storage Resources, e.g., store a history of sensor measurements, but are limited by the storage capacity of the Device.

As Network Resources run on a dedicated server in the network or in the “cloud”, they do not rely on special hardware that allows direct connection to the physical world. They rather provide enhanced Services that require more system resources than Devices typical for the IoT can provide. Such Resources can process data, for instance they can take sensor information as input and produce aggregated or more high-level information as output. Also, Network Resources can be storage Resources, which typically do not suffer from the limitations of their on-device counterparts. Storage Resources can store information produces by Resources and they can thus provide information about Physical Entities. This may include location and state-tracking information (history), static data (like product-type information), and many other properties. An example of a storage Resource is an EPCIS repository (Electronic Product Code Information Services (EPC 1.0.13)) that aggregates information about a large number of Physical Entities. Notice that also Human Users can update the information in a storage Resource, since not all known information about an entity always is, or even can be, provided by Devices.

7.3.3.3 Services

Services are a widely used concept in today's IT systems. According to (MacKenzie et al. 2006), "Services are the mechanism by which needs and capabilities are brought together". This definition is very broad, and the Service concept in the IoT Domain Model is covering this broad definition – but Services *are* restricted to technical Services implemented in software (in contrast to general, non-technical services that e.g. a lawyer or a consultant provides). As such, Services provide the link between the IoT aspects of a system and other, non-IoT specific parts of an information system, like e.g. various enterprise systems; IoT-related Services and non-IoT Services can be orchestrated together in order to form a complete system.

As it has been pointed out in (Martín 2012), IoT-related Services need to be explained in more detail: IoT Services provide well-defined and standardised interfaces, hiding the complexity of accessing a variety of heterogeneous Resources. The interaction with a Physical Entity can be accomplished via one or more Services associated with the corresponding Virtual Entity. This association becomes important in the process of look-up and discovery. An IoT Service can thus be defined as a type of Service enabling interactions with the real world.

According to (Martín 2012), IoT Services can be classified according by their level of abstraction:

- **Resource-level Services** expose the functionality, usually of a Device, by accessing its hosted Resources. These kinds of Services refer to a single Resource. In addition to exposing the Resource's functionality, they deal with quality aspects, such as dependability, security (e.g., access control), resilience (e.g., availability) and performance (e.g., scalability, timeliness). Resources can also be Network Resources, i.e. the Resources do not necessarily reside on a Device in the sense of the IoT Domain Model (normal computers are not regarded as IoT Devices by the IoT Domain Model), but can also be hosted somewhere else. The concrete location of where the Network Resource is situated is commonly abstracted away by the Service;

- **Virtual Entity-level *Services*** provide access to information at a Virtual Entity-level. They can be Services associated to a single Virtual Entity that give access to attributes for reading attribute information or for updating attributes in order to trigger associations. An alternative is to provide a common Virtual Entity-level *Service* with an interface for accessing attributes of different Virtual Entities, as, for instance, the NGSI Context Interface (NGSI 2010) provides for getting attribute information of the Virtual Entities;
- **Integrated Services** are the result of a Service composition of Resource-level or Virtual Entity-level *Services* as well as any combinations of both Service abstractions.

7.3.3.4 Identification of Physical Entities

In order to track and monitor Physical Entities, they have to be identified. There are basically two ways for how this can be done, as is very well described in (Furness 2009): Using either natural-feature identification (classified as "primary identification") or using some type of Tags or labels (classified as "secondary identification") that are attached to the Physical Entity.

Both means of identification are covered in the IoT Domain Model. Tags are modelled as Devices that explicitly identify a Physical Entity. Natural-feature identification can be modelled, for example, by using a camera – a kind of *Sensor* – that monitors the Physical Entity and an additional Resource that does the natural feature extraction (i.e. a dedicated software component). The result of the natural-feature extraction can be used as search term for looking up the corresponding Virtual Entity.

RFID Tags are a prominent example in IoT. As they come with their own electronic circuitry it seems quite natural to classify RFID Tags as Devices in terms of the IoT Domain Model. The case is less clear-cut regarding the classification of a barcode label, however. As pointed out elsewhere (Haller 2010), classifying a barcode label as a Device seems a little far-fetched; regarding it as a "natural feature" of the Physical Entity it is attached to, seems to be more appropriate. However, as with many modelling questions, this is a matter of taste – the IoT Domain Model is not prescribing which variant to use.

7.3.3.5 Context and Location

As the IoT pertains to the physical world, the characteristics of the physical world play an important role. All elements of the physical world are situated within a certain context, and location is an essential aspect of this context. All concepts in the IoT Domain Model that refer to elements of the physical world, i.e., Physical Entities, Devices, and Human Users inherently have a location. This location may or may not be known within the IoT system.

The location of a Physical Entity can be modelled as an attribute of a Virtual Entity. This location can then be provided through Resources. In the case of a stationary Physical Entity, the Resource providing the location can be an On-Device (storage) Resource, in the case of a mobile Physical Entity the Resource could be a positioning system like GPS, or a tracking system like existing indoor location systems.

7.4 Information Model

The IoT Information Model defines the structure (e.g. relations, attributes, services) of all the information for Virtual Entities on a conceptual level, see also Sects. 7.1.3.2.2, 7.1.3.3.1 and 7.1.3.3.3. The term information is used along to the definitions of the DIKW hierarchy (see Rowley 2007) where data is defined as pure values without relevant or useable context. Information adds the right context to data and offers answers to typical questions like who, what, where and when.

The description of the representation of the information (e.g. binary, XML, RDF etc.) and concrete implementations are not part of the IoT Information Model.

The IoT Information Model details the modelling of a Virtual Entity. The Virtual Entity (VirtualEntity) has attributes with a name (resp. Attribute and AttributeName) and a type (AttributeType) and one or more values (Value) to which meta-information (MetaData) can be associated. Important meta-information is, e.g., at what time a value was measured (i.e. time stamp), the location where a measurement took place, or the quality of the measurement. Metadata can itself contain additional metadata, e.g. the unit in which the metadata is measured. The association (Association) between a Virtual Entity and a Service is detailed in the sense that is pertains to a certain Attribute of the Virtual Entity. The serviceType can be set either to INFORMATION, if the *Service* provides the attribute value to be read or to ACTUATION, if the *Service* allows the Attribute value to be set, as resulting of a corresponding change in the physical world.

7.4.1 Definition of the IoT Information Model

The diagram in Fig. 7.8 shows the structure of the information that is handled and processed in an IoT System. The main aspects are represented by the elements VirtualEntity, ServiceDescription and Association. A Virtual Entity models a Physical Entity and ServiceDescription describes a Service that serves information about the Physical Entity itself or the environment. Through an Association, the connection between an Attribute of a Virtual Entity and the ServiceDescription is modelled, e.g. the Service acts as a "get" function for an Attribute value.

Every Virtual Entity needs to have a unique identifier (identifier) or entity type (entityType), defining the type of the Virtual Entity representation, e.g. a human, a

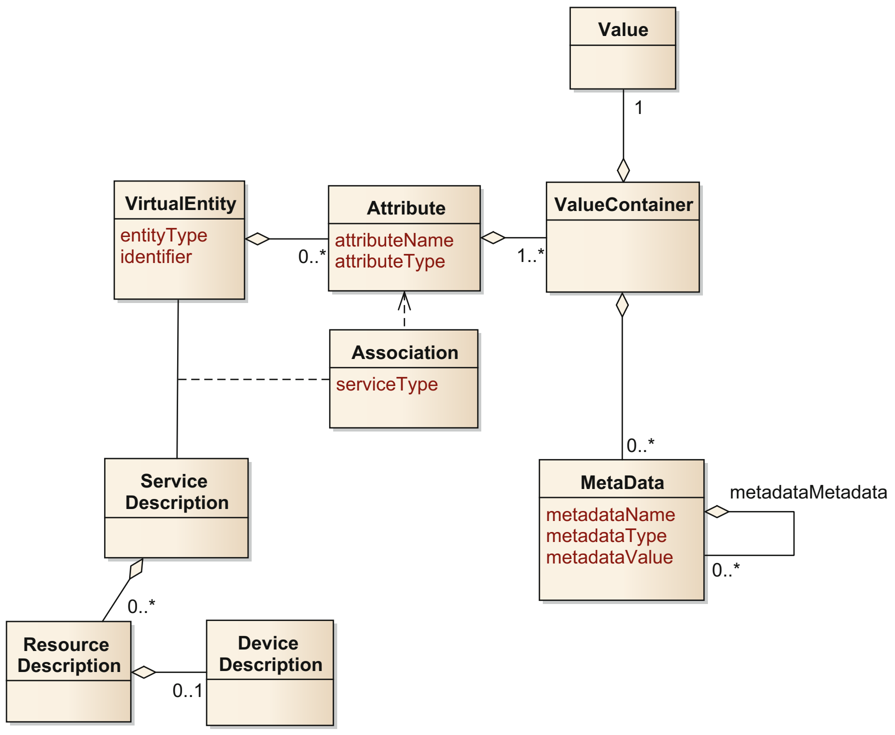


Fig. 7.8 IoT Information Model

car or a temperature sensor. Furthermore, a Virtual Entity can have zero to many different attributes (Attribute class in Fig. 7.8). The entityType of the VirtualEntity class may refer to concepts in an ontology that defines what attributes a Virtual Entity of this type has (see, for instance, [Group, W3C OWL]). Each Attribute has a name (attributeName), a type (attributeType), and one to *many* values (ValueContainer). The attributeType specifies the semantic type of an attribute, for example, that the value represents temperature. It can reference an ontology-concepts, e.g., qu:temperature taken from "Quantity Kinds and Units"-ontology (Lefort 2005). This way, one can for instance, model an attribute, e.g. a list of values, which itself has several values. Each ValueContainer groups one Value and zero to *many* metadata information units belonging to the given Value. The metadata can, for instance, be used to save the timestamp of the Value, or other quality parameters, such as accuracy or the unit of measurement. The Virtual Entity (VirtualEntity) is also connected to the ServiceDescription via the ServiceDescription-VirtualEntity association.

A ServiceDescription describes the relevant aspects of a Service, including its interface. Additionally, it may contain one (or more) ResourceDescription(s) describing a Resource whose functionality is exposed by the Service. The

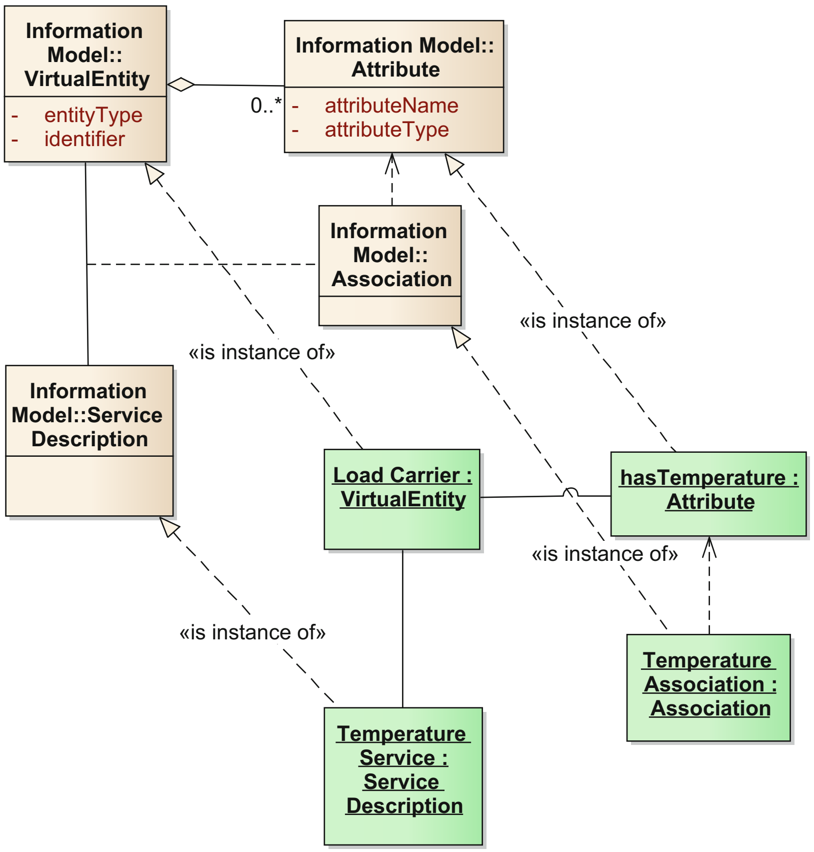

Information Model:: VirtualEntity
- entityType
- identifier
Information Model:: Attribute
- attributeName
- attributeType
0..*
Information Model:: Association
Information Model::Service Description
«is instance of»
«is instance of»
Load Carrier : VirtualEntity
hasTemperature : Attribute
«is instance of»
«is instance of»
Temperature Association : Association
Temperature Service : Service Description

Fig. 7.9 Illustrating example for IoT Information Model

ResourceDescription in turn may contain information about the Device on which the Resource is hosted.

7.4.2 Modelling of Example Scenario

The IoT Information Model is a meta-model that defines the structure of key aspects of the information being managed in an IoT system. Therefore, unlike the Domain Model (see the recurring example in Sect. 7.1.3.1), it would typically not be directly instantiated (see information view Sect. 8.2.3 and the related Design Choices in Chap. 6 for this purpose). Nevertheless, for illustration purposes, we sketch here how the information relevant for our example scenario from Sect. 4.2 could be modelled (Fig. 7.9).

The element of interest for which we can get some information is the Load Carrier, which is therefore digitally represented by a Virtual Entity. Here, we show how the temperature aspect of the Physical Entity is modelled by the hasTemperature attribute of the Virtual Entity. This Figure also features a description of the service that is used to measure this temperature. What is finally needed is the connection between the hasTemperature attribute and the service that can provide its value. This is achieved by the Temperature Association.

7.4.3 Relation of Information Model to Domain Model

The IoT Information Model models all the concepts of the Domain Model that are to be explicitly represented and manipulated in the digital world. Additionally, the IoT Information Model models relations between these concepts. The IoT Information Model is a meta-model that provides a structure for the information being handled by IoT Systems. This structure provides the basis for all aspects of the system that deal with the representation, gathering, processing, storage and retrieval of information and as such is used as a basis for defining the functional interfaces of the IoT system.

Figure 7.10 shows the relation between the Domain Model concepts and the IoT Information Model elements. The main Domain Model concepts that are explicitly represented in an IoT system are the Virtual Entity and the service. The latter also comprises aspects of the Resource and the Device. As the Virtual Entity is the representation of the Physical Entity in the digital world, there is no other representation of the Physical Entity in the IoT Information Model.

7.4.4 Other Information-Related Models in IoT-A

Throughout IoT-A several other information- related models exist. Most of them are defined in the technical work packages WP2, WP3, WP4 and WP5. More information can be found in the respective deliverables (see below).

- **Entity model:** The Entity Model specifies which attributes and features of real word objects are represented by the virtual counterpart, i.e. the Virtual Entity of the respective Physical Entity. For every attribute specified in the entity model, services can be found that are able to either provide information about the attribute (sensing) or manipulate it, leading to an effect in the real world (actuating). More information about the entity model can be found in Sect. 7.1.3.2.2.
- **Resource model:** The Resource Model contains the information that is essential to identify Resources by a unique identifier and to classify Resources by their type, like sensor, actuator, processor or tag. Furthermore the model specifies the

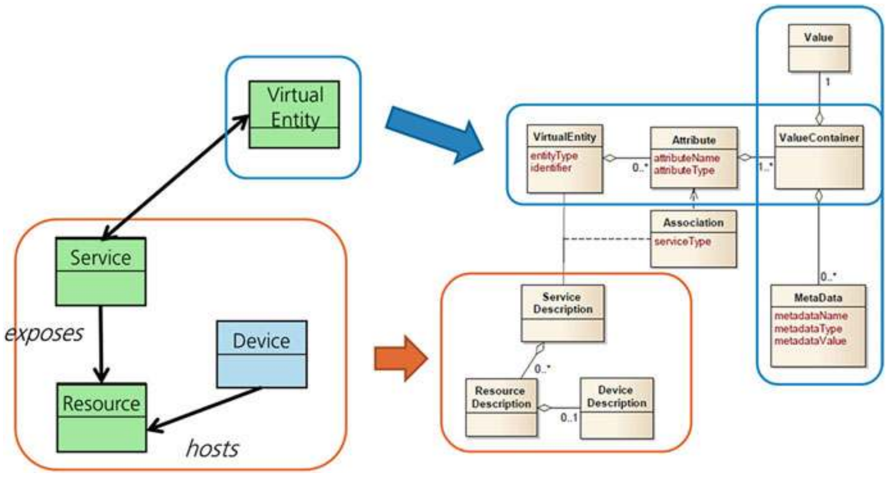


Fig. 7.10 Relation between the core concepts of the IoT Domain Model and IoT Information Model

geographic location of the Resource, the Device the Resource is hosted on (if so) as well as the IoT Services the Resource is exposed through. More information can be found in (Martin 2012) Sect. 3.3.

- **Service description model:** Services provide access to Resources and are used to access information or to control Physical Entities. An IoT Service accesses IoT Resources in order to provide information about attributes of entities or manipulates them leading to an effect in the real world. A Service Description describes a Service, using for instance a service description language such as USDL.[1] For more information see (Martin 2012) Sect. 4.6.3.
- **Event Model:** Event models are quite essential in today's IoT architectures, e.g. in the EPCglobal Information Services. Normally events are used to track dynamic changes in a (software) system, showing who or what has triggered it and when, where and why the change occurred. Event representation and processing is specified in Sect. 4.2 of (Voelksen 2013).

7.5 Functional Model

7.5.1 Functional Decomposition

In the IoT-A project, *Functional Decomposition* (FD) refers to the process by which the different *Functional Components* (FC) that make up the IoT ARM are identified and related to one another.

[1] http://www.internet-of-services.com/.

The main purpose of Functional Decomposition is, on the one hand, to break up the complexity of a system compliant to the IoT ARM in smaller and more manageable parts, and to understand and illustrate their relationship on the other hand.

Additionally, Functional Decomposition produces a superset of functionalities that can be used to build any IoT system. The output of Functional Decomposition is described in this document at two levels of abstraction:

- The Functional Model (purpose of this section);
- The Functional View (presented in Sect. 8.2.2).

The definition of the Functional Model is derived by applying the definition of a Reference Model found in (MacKenzie et al. 2006) to Functional Decomposition: "The Functional Model is an abstract framework for understanding the main *Functionality Groups* (FG) and their interactions. This framework defines the common semantics of the main functionalities and will be used for the development of IoT-A compliant Functional Views."

The definition contains the following concepts that need to be explained in more detail:

- **Abstract:** The Functional Model is not directly tied to a certain technology, application domain, or implementation. It does not explain what the different Functional Components are that make up a certain Functionality Group;
- **Functionality Groups and their interactions:** The Functional Model contains both the Functionality Groups and the interaction between those parts. A list of the Functionality Groups alone would not be enough to make up the Functional Model. Both the Functionality Groups and their interaction are mandatory;
- **Functional View:** The Functional View describes the system's runtime Functional Components, including the components' responsibilities, their default functions, their interfaces, and their primary interactions. The Functional View is derived from the Functional Model on the one hand and from the Unified Requirements on the other hand. Note that various Functional Views could be derived from the Functional Model. See also Sect. 8.2.2 for more detailed information on the functional view.

7.5.2 *Functional Model Diagram*

The Functional Model diagram is depicted in Fig. 7.11 and was derived as follows:

- From the main abstractions identified in the Domain Model (Virtual Entities, Devices, Resources and Users) the "Application", "Virtual Entity", "IoT Service" and "Device" FGs are derived;
- With regards to the plethora of communication technologies that the IoT ARM needs to support, the need for a "Communication" FG is identified;

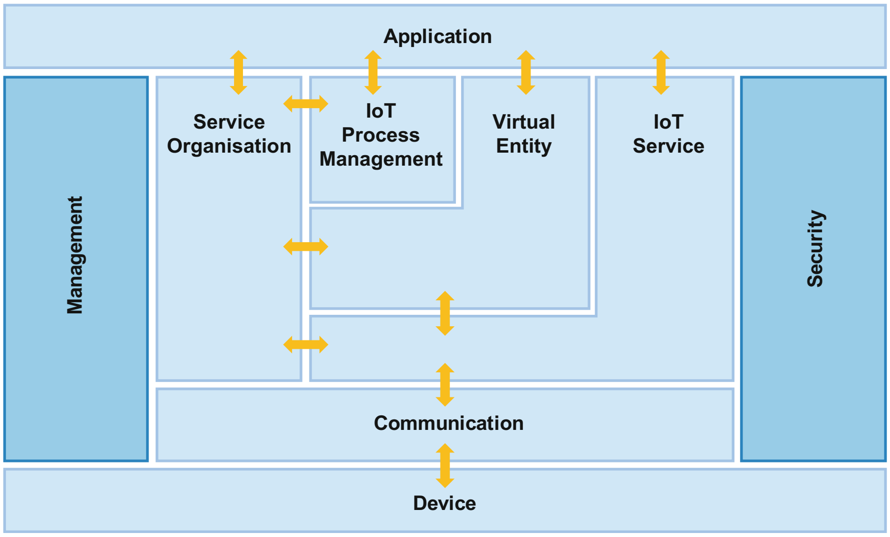


Fig. 7.11 Functional Model

- Requirements expressed by stakeholders regarding the possibility to build services and applications on top of the IoT are covered by the "IoT Process Management" and "Service Organisation" FGs;
- To address consistently the concern expressed about IoT Trust, Security and Privacy, the need for a "Security" transversal FG is identified;
- Finally, the "Management" transversal FG is required for the management of and/or interaction between the functionality groups.

The Functional Model contains seven longitudinal Functionality Groups (light blue) complemented by two transversal Functionality Groups (Management and Security, dark blue). These transversal groups provide functionalities that are required by each of the longitudinal groups. The policies governing the transversal groups will not only be applied to the groups themselves, but do also pertain to the longitudinal groups.

As an example: for a security policy to be effective, it must ensure that there is no functionality provided by a component that would circumvent the policy and provide unauthorised access.

Next, the interactions between the FGs are shown. As can be seen from Fig. 7.11, the Functional Model is a hierarchical model and the main interactions between the FG's are depicted with orange arrows. Since the transversal FGs (Management & Security) interface with most of the other FGs, their interactions with the other FG's are not explicitly depicted.

In the remainder of this section, each of the FGs will be described in more detail (with exception of the Application and Device FGs, since trying to capture their properties would be so generic that they do not add any value).

7.5.2.1 IoT Process Management

The IoT Process Management FG relates to the conceptual integration of (business) process management systems with the IoT ARM. The overall aim of this FG is to provide the functional concepts necessary to conceptually integrate the idiosyncrasies of the IoT world into traditional (business) processes. By so doing, enterprises can effectively utilise IoT sub-systems adhering to common standards and best practices, thus avoiding the overhead and costs of isolated and proprietary "Intranet-of-Things" island solutions.

In the IoT-A project, the IoT Process Management FG is addressed by WP 2. The IoT Process Management FG provides additions and extensions to industry standards, for instance BPMN 2.0. The additions feature IoT-specific aspects of (business) processes, such as the reliability or accountability of sensor data providing information about Virtual Entities or the required processing capabilities of Devices hosting certain Resources relevant for the real world. Applications that interact with the IoT Process Management FG via IoT-augmented process models can effectively be shielded from IoT-specific details of lower layers of the functional model, which greatly reduces integration costs and thus contributes to an increased adoption of IoT-A based IoT systems (Meyer et al. 2011).

One important aspect of IoT Process Management is its inherent closeness to enterprise systems. As it was already introduced in the IoT Domain Model Sect. 7.1.3, the IoT Process Management FG is where the business objects and processes are combined with the world of IoT, and especially here the modelling of processes must take into account not only the idiosyncrasies of the IoT domain, but also the specificities of the underlying business domain. The different roles of the business objects and users will be defined here. Again, as discussed in the IoT Domain Model section, we do not prescribe a specific taxonomy here. However, for pedagogical purposes we illustrate how this taxonomy looks like in the context of the RM-ODP context Enterprise View (see the discussion about RM-ODP (Raymond 1995) and roles in IoT Domain Model Sect. 7.1.3.2.2).

- Permission: what can be done? For instance a self-regulating ventilation system can be started by a central control system;
- Prohibition: what must not be done? For instance the ventilation system may not be shut down in its entirety if the outside temperature is above a pre-defined value and if humans are present in the building;
- Obligations: the central control system needs to save recorded environmental parameters for each room in the entire building (temperature, humidity, ventilation settings). Such records can, for instance, be required by national occupational-health laws.

When it comes to the practical realisation of the process management, these different policies will come into play when the respective business processes are modelled. In Chap. 5 we pick up the notion of enterprise views and illustrate how they factor into the requirements process.

The IoT Process Management FG is conceptually closely related to the Service Organisation FG and acts as a proxy to applications that integrate an IoT-A-compliant IoT system. Naturally, the IoT Process Management FG has a dependency on the Service Organisation FG, as a central concept in the execution of (business) processes is the finding, binding, and invoking of Services that are used for each process step. The IoT Process Management FG therefore relies on Service Organisation to map the abstract process definitions to more concrete Service invocations.

Applications can utilise the tools and interfaces defined for the IoT Process Management FG in order to stay on the (abstract) conceptual level of a (business) process, while, at the same time, making use of IoT-related functionality without the necessity of dealing with the complexities of IoT Services. In this respect, the IoT Process Management FG provides conceptual interfaces to the IoT ARM, that are alternatives to the more concrete Virtual Entity FG and Service Organisation FG interfaces.

7.5.2.2 Service Organisation

The Service Organisation FG is a central Functionality Group that acts as a communication hub between several other Functionality Groups. Since the primary concept of communication within the IoT ARM is the notion of the Service (see Domain Model Sect. 7.1.3), the Service Organisation FG is used for composing and orchestrating Services of different levels of abstraction. Within the IoT Reference Architecture, it effectively links the Service requests from high level FGs such as the IoT Process Management FG, or even external applications, to basic services that expose Resources (see Domain Model Sect. 7.1.3) (such as services hosted on a WSN gateway), and enables the association of entities with these services by utilising the Virtual Entity FG, so that a translation of high-level requests dealing with properties of entities (e.g., "give me please the temperature in the room 123") down to the concrete IoT services (e.g., "sensor service XYZ") can be realised. In order to allow for querying Virtual Entities or IoT Services that are associated with these entities, the Service Organisation FG is responsible for resolving and orchestrating IoT Services and also deal with the composition and choreography of Services. Service Composition is a central concept within the architecture, since IoT Services are very frequently capable of rather limited functionality due to the constraints in computing power and battery life that are typical for WS&ANs or embedded Devices comprising the IoT. Service Composition then helps combining multiple of such basic Services in order to answer requests at a higher level of Service abstraction (e.g. the combination of a humidity sensing Service and a temperature Service could serve as input for an air-conditioning). Service Choreography is a concept that supports brokerage of Services so that Services can subscribe to other Services available in the system.

As discussed in the previous section, the Service Organisation FG is closely tied to the IoT Process Management FG, since the Service Organisation FG enables (business) processes or external applications to find and bind Services that can be used to execute process steps, or to be integrated in other ways with external applications. The Service Organisation FG acts as an essential enabler for the IoT Process Management FG. The Virtual Entities specified during the process modelling phase are resolved and bound to IoT Service FG needed for process execution.

7.5.2.3 Virtual Entity and IoT Service

The Virtual Entity and IoT Service FGs include functions that relate to interactions on the Virtual-Entity and IoT-Service abstraction levels, respectively. Figure 7.12 shows the abstraction levels and how they are related. On the left side of Fig. 7.12, the physical world is depicted. In the physical world, there are a number of Sensors and Actuators that capture and facilitate the change of certain aspects of the physical world. The Resources associated to the Sensors and Actuators are exposed as IoT Services on the IoT Service level. Example interactions between applications and the IoT system on this abstraction level are “Give me the value of Sensor 456” or “Set Actuator 867 to On”. Applications can only interact with these Services in a meaningful way, if they already know the semantics of the values, e.g. if Sensor 456 returns the value 20, the application has to be programmed or configured in such a way that it knows that this is the outdoor temperature of the car of interest, e.g. Car MXD – 123. So, on this level no semantics is encoded in the information itself, nor does the IoT system have this information, it has to be a-priori shared between the Sensor and the application.

Whereas interaction on the IoT Service level is useful for a certain set of applications that are programmed or configured for a specific environment, there is another set of applications that wants to opportunistically use suitable Services in a possibly changing environment. For these types of applications, and especially the Human Users of such applications, the Virtual Entity level models higher-level aspects of the physical world, and these aspects can be used for discovering Services. Examples for interactions between applications and the IoT system on this abstraction level are “Give me the outdoor temperature of Car MXD – 123” or “Set lock of Car MXD – 123 to locked”. To support the interactions on the Virtual Entity level, the relation between IoT Services and Virtual Entities needs to be modelled, which is done in form of associations. For example, the association will contain the information that the outdoor temperature of Car MXD – 123 is provided by Sensor 456. Associations between Virtual Entities and IoT Services are modelled in the Information Model (Sect. 7.1.4).

Virtual Entity

The Virtual Entity FG contains functions for interacting with the IoT System on the basis of VEs, as well as functionalities for discovering and looking up Services that can provide information about VEs, or which allow the interaction with VEs.

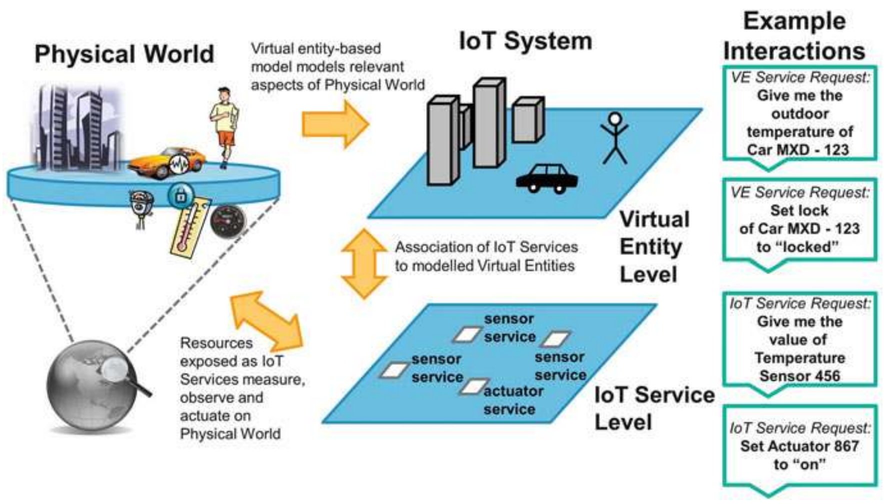

Physical World
Virtual entity-based model models relevant aspects of Physical World
IoT System
Example Interactions
VE Service Request: Give me the outdoor temperature of Car MXD - 123
VE Service Request: Set lock of Car MXD - 123 to "locked"
Virtual Entity Level
Association of IoT Services to modelled Virtual Entities
IoT Service Request: Give me the value of Temperature Sensor 456
IoT Service Request: Set Actuator 867 to "on"
sensor service
sensor service
sensor service
actuator service
IoT Service Level
Resources exposed as IoT Services measure, observe and actuate on Physical World

Fig. 7.12 IoT-Service and Virtual-Entity abstraction levels

Furthermore, it contains all the functionality needed for managing associations, as well as dynamically finding new associations and monitoring their validity. This need can be triggered by the mobility of Physical Entities represented by the Virtual Entities and/or Devices.

IoT Service

The IoT Service Functional Group contains IoT Services as well as functionalities for discovery, look-up, and name resolution of IoT Services.

7.5.2.4 Communication

The Communication FG abstracts the variety of interaction schemes derived from the many technologies (Device FG) belonging to IoT systems and provides a common interface to the IoT Service FG. It provides a simple interface for instantiating and for managing high-level information flow. In particular, the following aspects are taken into account: starting from the top layers of the ISO/OSI model it considers data representation, end to end path information, addressing issues (i.e. Locator/ID split), network management and device specific features.

The Communication FG can be customised according to the different requirements defined in the Unified Requirements list and, in particular, those related to communication specified within WP3. For instance, integrity and security can be enforced exploiting many different signature and encryption schemes at various ISO/OSI layers; reliability is achieved either by means of link layer acknowledgements or end to end error correction schemes at the upper layers; quality of service is realised by providing queue management techniques; finally, in order to account for communication between different technologies, protocol translation and context passing functionalities are described.

7.5.2.5 Management

The Management FG combines all functionalities that are needed to govern an IoT system. The need for management can be traced back to at least four high-level system goals (Pras 1995):

- Cost reduction;
- Attending unexpected usage issues;
- Fault handling; and
- Flexibility.

Cost reduction: In order to control the cost of a system, it is designed for a maximum amount of users and/or use cases. "A way for the designer to deal with the requirements of multiple groups of users is to abstract from the differences in [the] requirements and [to] parameterise the design" (Pras 1995). Upon commissioning or start-up of the system, these parameters will be initialised by the Management FG.

Attending unexpected events: The IoT system is based on an incomplete model of reality – as literally all systems are. For example, even for the same type of user, unforeseen activity patterns in the physical world and thus unforeseen usage may arise may arise. For instance, errors are introduced into the system through explicit, erroneous management directives (Harrisburg, Chernobyl). Another example is that Devices can suddenly just die. The latter is most likely to become prevalent in the IoT, since the cost margins for IoT equipment and thus their reliability can be much lower than that for traditional telecommunications equipment (back-bone routers, etc.). The management FG can provide strategies and actions for the mitigation of impacts from unforeseen situations. Such impacts can be link failure, queue overload, etc. In order to better adapt to new situations, it is of course paramount that the Management FG has a good overview of the system state. To that end the management system provides supports collection.

Fault handling: This goal addresses the unpredictability of the future behaviour of the system itself. This is of special interest in complex IoT systems and also in IoT systems in which, for instance, the devices in an IoT system do not provide a model for their behaviour. The measures implied by this goal are:

- Prediction of potential failures;
- Detection of existing failures;
- Reduction of the effects of failures;
- Repair.

The first three measures can be achieved by comparing the current behaviour of system components with previous and/or expected behaviour.

Flexibility: The design of a system is based on use-case requirements. However, these use-case requirements are not static. Instead of designing a new system every time the requirements change, some flexibility should be built into the system. Due to this flexibility, the Management FG of the IoT system will be able to react to changes in the user requirements. This can take place during boot up, commissioning or also at run time.

All of the above goals rely on shared common functionality and repositories, as, for instance, a state repository. Other functionalities are:

- Management of the membership and accompanying information of a given entity to the IoT system. Such entity may be a Functional Component (FC), a Virtual Entity, an IoT Service, an application, a Device. The information considered may cover ownership, administrative domain, capabilities, rules, and rights;
- Retrieval of the list of members pertaining to a given property such as the ownership/administrative domain;

Finally, some more examples for the above goals are provided:

- Enforcing rules attached to the usage of a certain entity e.g.
  - *Attending unexpected events:* A service needs temperature measurements every microsecond, but the rule file for the associated device says: maximum measurement frequency of this device is 100 Hz. The rule file also might say: no continuous operation of said device for more than 1 h (due to energy constraints);
  - *Fault handling:* A service wants to run a business process that would consume all IoT services and the VE lookup for more than a day. An example for this is a query for the geo-location of all temperature Sensors on planet Earth. The rule file may contain instructions about how many resources can be consumed by an application;
  - *Cost reduction:* Logging entity usage by a user for further processing (e.g. billing).

Besides the above, "traditional" goals of management, the Management FG also needs to address needs that arise when IoT systems can actuate and/or if the they are embedded in critical infrastructure. Examples for such situations are

- Bringing the entire system to an emergency stop, for instance a train;
- Turning the entire system into a sleep/energy-saving mode in order to relax to load on a failing Smart Grid.

7.5.2.6 Security

The Security Functionality Group (Security FG) is responsible for ensuring the security and privacy of IoT-A-compliant systems. It is in charge of handling the initial registration of a client to the system in a secure manner. This ensures that only legitimate clients may access services provided by the IoT infrastructure. The Security FG is also in charge of protecting private parameters of users. This is achieved by providing anonymity (ensuring that the user's identity remains confidential when she/he/it accesses a Resource or a service) and "unlink-ability" (ensuring that the user may make multiple uses of Resources or services without an attacker being able to establish links between those uses). This privacy support relies on fine-tuned identity management, which is able to assign various pseudo-random identifiers to a single user.

The Security FG also ensures that legitimate interaction occurs between peers that are statically authorised to interact with each other, or that are trusted by each other. This is achieved through the use of dedicated authorisation functions or through the reliance on a trust – and-reputation model, which is able to identify trustworthy peers in a privacy-capable and highly mutable architecture.

Finally, the Security FG enables secure communications between peers by managing the establishment of integrity and confidentiality features between two entities lacking initial knowledge of each other.

7.6 Communication Model

The IoT Communication Model aims at defining the main communication paradigms for connecting elements, as defined in the IoT Domain Model. We provide a reference set of communication rules to build interoperable stacks, together with insights about the main interactions among the elements of the IoT Domain Model. We propose a Communication Model that leverages on the ISO OSI 7-layer model for networks and aims at highlighting those peculiar aspects inherent to the interoperation among different stacks, which we will call in what follows, interoperability features. Further, the application of communication schemes, such as application layer gateways, transparent proxy, network virtualization, etc., to different IoT network types is discussed.

In particular, with reference to our Read Thread example of Sect. 4.2, the IoT Communication Model can be used to model how the monitoring Sensors of the truck can seamlessly interact with Ted's phone and how it can communicate with the store enterprise system.

The IoT Communication Model has multiple usages. For instance, it can guide the definition of the Communication Functional Components from which the Communication Functional Group is composed of. Finally, it can be used to derive the Communication best practices, as depicted in the following pictures (Fig. 7.13):

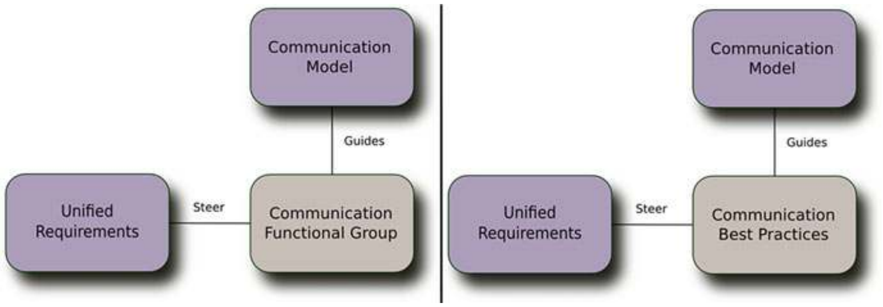


Fig. 7.13 IoT Communication Model usages: (*left*) using the CM together with the Unified Requirements to define the Communication FG; (*right*) deriving Communication Best Practices thanks to the CM and the Unified Requirements

7.6.1 IoT Domain Model Element Communications

For the IoT Communication Model, it is important to identify the communication-system elements and/or the communicating Users among those defined in the IoT Domain Model. One, if not the main peculiarity of the IoT is that Users can belong to many disjoint categories: Human Users, Services or Active Digital Artefacts. While the same picture is emerging in today's Internet usage, the percentage of human-invoked communication will be even lower in the IoT. Moreover, entities can be physical, digital, or virtual. While a Physical Entity cannot directly take part in communication, it can exploit Services associated to its virtual counterpart.

The communication between these users needs to support different paradigms: unicast is the mandatory solution for one-to-one connectivity. However, multicast and anycast are needed for fulfilling many other IoT-application requirements, such as data collection and information dissemination, etc.

With reference to our "Red Thread" and the IoT Domain Model section, the main communicating elements are: the Mote Runner Node (Device), the Alarm Service (Service), the AndroidApp (Active Digital Artefact) and Ted (Human User).

This section provides insight and guidance on the interactions between elements of the IoT Domain Model. In particular, per possible communicating entity pair, a discussion about the relevant layer of the IoT Communication Model will be provided.

7.6.1.1 User-Service / Service-Service Interactions

As shown in Fig. 7.14, the IoT Domain Model entities involved in this interaction are mainly two: User and Service (circled in solid red lines). For instance, in our recurring example this interaction models the truck driver, Ted, who needs to interact with the AndroidApp in order to receive alarms. However, a Service may

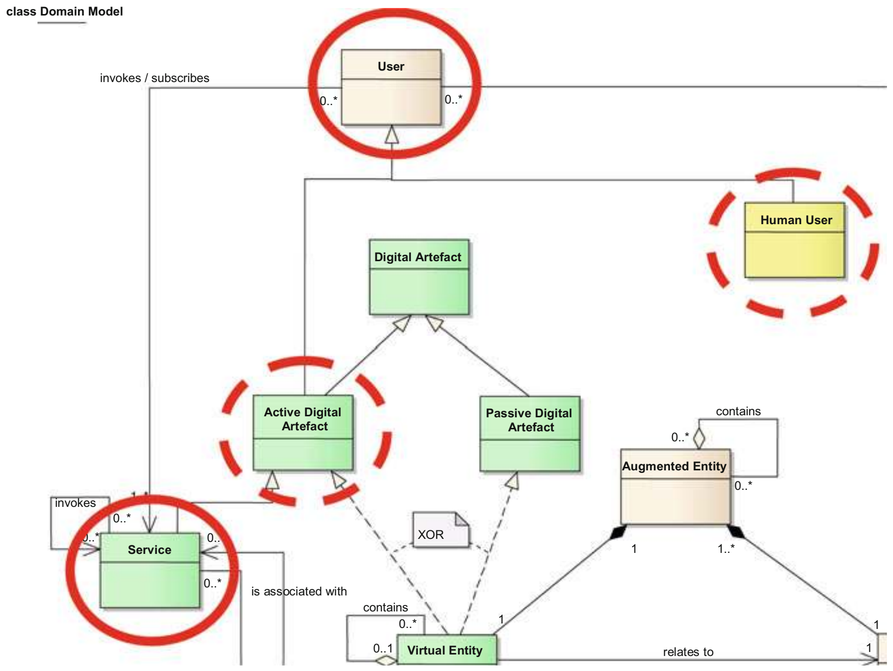


Fig. 7.14 DM entities involved in a User-Service / Service-Service interaction (zoom of the whole IoT Domain Model in Fig. 7.7)

also assume the user role when invoking another Service, thus Users can either be Human User or Active Digital Artefacts.

This interaction is straightforward, as it is identical to typical Internet interactions between Users and Services. In fact, in most of the application scenario the User-Service connection can be established using standard Internet protocols.

However, two main exceptions to this general assumption apply when two Services communicate one to each other and one or both of the communicating elements belong to a constrained network.

The latter case, which applies when Services are deployed on constrained Devices such as Sensor nodes and when the User of a given Service is deployed on a constrained Device, requires for the use of constrained communication protocols (see Rossi 2012). Finally, when the two elements belong to different sub-networks, gateway(s) and/or proxy(ies) must be deployed for ensuring successful end-to-end communication. To this extent, as a general rule, if a Service is constrained, or if it needs to provide access to constrained Users, it must be accessible with constrained protocols (e.g., 6LoWPAN, UDP, CoAP, etc.).

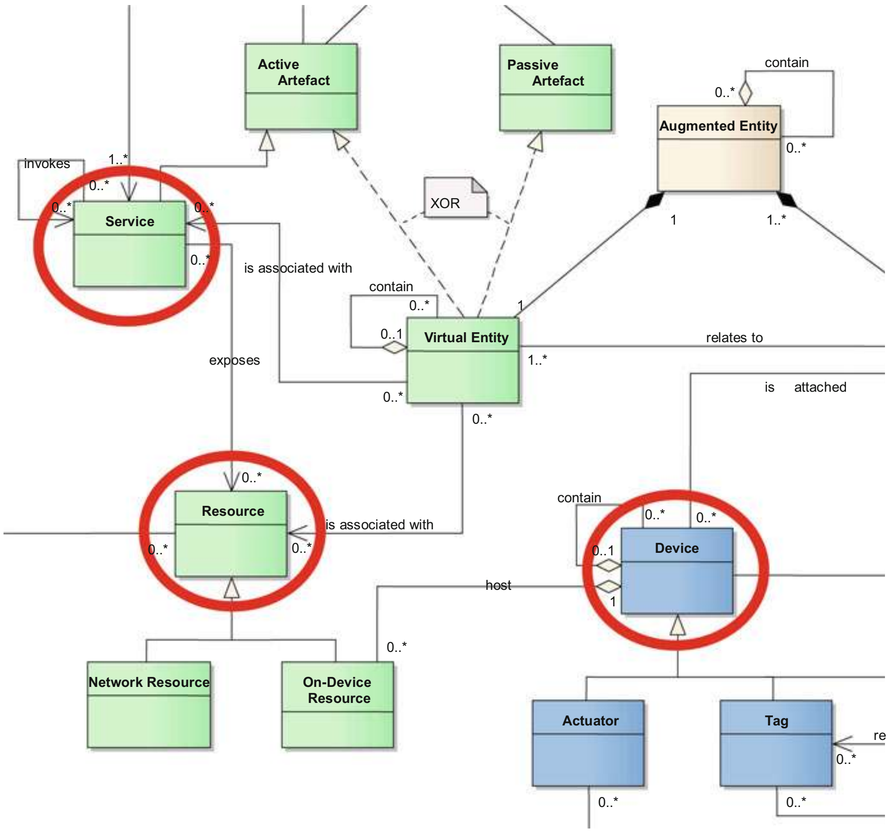


Fig. 7.15 IoT DM entities involved in a Service / Resource / Device interaction (This is a zoom of the complete IoT DM highlighting the interested parts, see also Fig. 7.7)

7.6.1.2 Service / Resource / Device Interactions

Figure 7.15 illustrates the entities of the IoT Domain Model involved in the interactions among Services, Resources and Devices. These interactions can be exemplified with the communication among the Alarm Service, the Alarm Resource and the Mote Runner Node of the recurring example. This Figure also illustrates the interconnections of these entities.

The complexity of this interaction is due to variety of different properties that a Device can have; in particular, a Device can be as simple and limited as a Tag and as complex and powerful as a server (Tag Terminal Type (TT3) and unconstrained Terminal Type (TT1), respectively, in (Rossi 2012)). In fact, while powerful Devices can easily support the needed software to host and access Services and to expose the needed Resources for other Services to interact with, simpler Devices may only be able to provide access to their own Resources and the simplest may even not be powerful enough even to support this. In the latter two cases, Resources and Services must be provided somewhere else in the network, by some other more powerful Device(s).

Thus the IoT Communication Model helps to model and to analyse how constrained Devices can actively participate in an IoT-A compliant communication and to study possible solutions, such as the usage of application layer gateways, to integrate legacy technologies.

7.6.2 Communication Interoperability Aspects

The model we are going to propose in this section takes its roots from the ISO/OSI (ISO 1994) stack, the US Department of Defense 4 layer model (DoD4) (Darpa 1970) and, the Internet stack, but it puts its focus on IoT specific features and issues. All the previous models have a great value, going beyond any discussion, but simply they have not been conceived with the IoT issues and features in mind.

In Fig. 7.16 we can see the Internet and the DoD4 stacks. It is evident how they map onto each other, thus in what follows we will address the 4 layers Internet model only.

The 4-layer Internet stack abstracts from the underlying Physical Layer; in fact its lowest layer is represented by the Link Layer. This choice is indeed the right one for the Internet, as the Link Layer is not visible from the Application Layer, and the same can be applied to fully homogeneous networks, since applications can be totally agnostic of the underlined common physical technology. However, the Physical Layer rises to a great importance when talking about the IoT; in fact the IoT is characterized by a high heterogeneity of hardware and technologies and the necessity of making different system interoperable. Moreover, this is a clear statement on the fact that IP is conceived in order to be implemented on top of any hardware networking technologies, while in the IoT there exist technologies that do not dispose of the needed resources to manage a complete IP compliant communication. Thus, solutions such as 6LoWPAN, are needed to extend IP communication to constrained networks.

Moreover the main objective of the 4-layer Internet model is to let Internet applications communicate, having intermediate devices understanding the communication at IP level, without meddling with upper layers. This model is wonderful in its simplicity, but this simplicity is one of the reasons why it is unsuitable for the IoT, since it is not able to address the aforementioned interoperability issues.

Obviously this dates back to the beginning of the Internet, when developing an Application Protocol for each application was best practice. While in principle that is a reasonable approach, even in the current Internet we can perceive how it is misleading. We are not here to discuss pros and cons of developing an Application

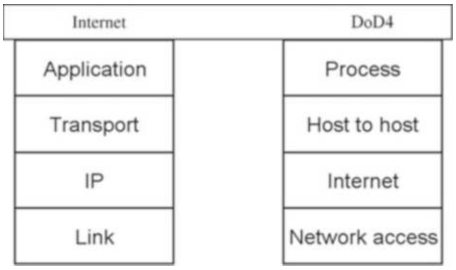


Fig. 7.16 Four layers Internet stack (*left*) and DoD4 stack (*right*)

Protocol for each application but we have just to notice this is not a common practice anymore, with the majority of applications leveraging on HTTP or even on more specific HTTP constructs like REST protocols. So nowadays it is crucial to have a distinction between Applications and the Application Protocols.

Another major issue of the 4 layers Internet model arises from the lack of expression for the so called security layers, the two major examples being SSL (IETF 2011)/TLS (IETF 2008) and IPsec (IETF 1998).

The main reference point for communication system developers is the ISO/OSI stack, and, although its validity as an implementation guideline is out of question, it fails to depict the overall complexity of IoT systems as it is meant to represent single technology stacks.

After the considerations on the model discussed so far we felt necessary a different approach for highlighting the peculiar features of IoT communication, which are not directly evident using the ISO/OSI model alone.

The model, as depicted in Fig. 7.17 on the left-hand side, stresses the relevance of the wide range of interoperability aspects that characterise IoT systems. In fact, instead of focusing on a specific realisation of the communication stack, the IoT Communication Model provides a transversal approach from which one or more communication stacks can be derived: in fact a single interoperability aspect can be used to describe the interactions of stacks belonging to different communicating systems. Once a system is modelled according to the IoT Communication Model it is easy to derive a set of ISO/OSI interoperable stacks in order to provide the needed interoperability features.

Below, the different interoperability aspects are described:

- **Physical aspect**: This interoperability aspect concerns the physical characteristics of the communication technologies used in the system. It is similar to the OSI Physical Layer. This is necessary in order to neither exclude any available technology, and to prevent emerging solutions from being integrated into the Reference Model. This aspect does not force the adoption of any specific technology, but it uses the adopted technologies as a base to model the remaining of the system. In fact, as per the recurring example the Mote Runner Node can communicate using some low-power radio transceiver such as ZigBee, while the AndroidApp can be hosted in an IoT-Phone connected to the Internet either via WiFi or 3G cellular networks;

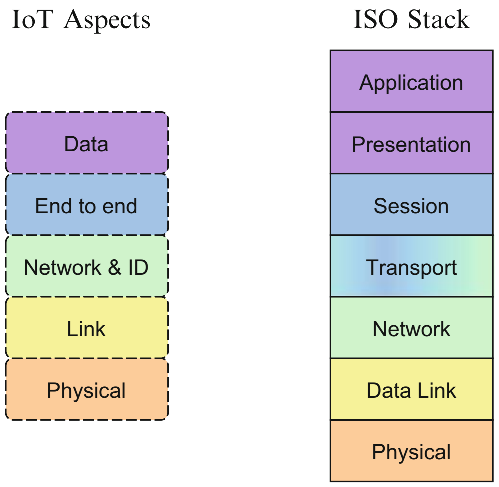


Fig. 7.17 The interoperability aspects of the IoT Communication Model (*left*) compared to the ISO/OSI communication stack (*right*): dashed lines have been used for the IoT aspects to make them graphically different from stack layers

- **Link aspect**: In order to address the heterogeneousness of networking technologies represented in the IoT field, the Link aspect requires special attention. In fact, most networks implement similar, but customised communication schemes and security solutions. In order for IoT systems to achieve full interoperability, as well as the support of heterogeneous technologies and a comprehensive security framework, this layer must support solution diversity. At the same time, it needs to provide upper layers with standardised capabilities and interfaces. Therefore, this layer needs to abstract a large variety of functionalities, enabling direct communication. IoT systems do not have to restrict the selection among data link layers, but must enable their coexistence;
- **Network and ID aspect**: This interoperability aspect combines two communication aspects: *networking*, which provides the same functionalities as the correspondent OSI layer; and *identifiers*, which are provided using resolution functionalities between locators and IDs. In order to support global manageability, interoperability, and scalability, this aspect needs to provide a common communication paradigm for every possible networking solution. This is the narrow waist for the Internet of Things. The difference between identifiers (unique descriptors of the Digital Artefact; either active or passive), and locators (descriptors of the position of a given IoT element in the network), is the first convergence point in the IoT Communication Model. Thus, this interoperability aspect is in charge of making any two systems addressable from one another notwithstanding the particular technologies they are adopting. In the case of our recurring example the AndroidApp must be able to receive alarms generated by the alarm Service, which in turns, must receive information from the Mote Runner Device: in order for this to be possible the system must ensure that the correct identifiers are supported by all the communicating technologies or can be resolved via appropriate methods;

- **End-to-end aspect**: this aspect takes care of reliability, transport issues, translation functionalities, proxies/gateways support and parameter configuration when the communication crosses different networking environments. By providing additional interoperability aspects on top of those of the Network and ID aspect, this aspect provides the final component for achieving a global M2M communication model. Connections are also part of the end-to-end scope. Also, Application Layer aspects are taken care of here. Moreover Application Protocols in the IoT tend to embed confirmation messages, and congestion control techniques require being more complex than what is achievable in the Transport Layer in the legacy models. With reference to the recurring example, this aspect will take care of modelling the overall communication between the Alarm Service and the Mote Runner Node and between the AndroidApp and the Alarm Service;
- **Data aspect**: the topmost aspect of the IoT Communication Model is related to data definitions and transfers. While the Information Model provides a high-level description for data of IoT systems, the purpose of this aspect is to model data exchange between any two actors in the IoT. As described in the IoT Information Model (see Sect. 7.4), data exchanged in IoT can adopt many different representations, ranging from raw data to complex structures where meta-information is added to provide context specific links. Finally, to make this possible, the data aspect needs to model the following characteristics (Rossi 2013): (1) capability of providing structured attributes for data description; (2) capability of being translated (possibly by compression/decompression) the one to each other, e.g. CoAP is translatable to HTTP by decompression or XML is translatable to EXI by compression, IPv4 is translatable to IPv6 by mapping; (3) constrained device support. For instance, in the recurring example, the raw data produced by the Mote Runner Sensors shall be converted into machine-readable formats in order for the Alarm Service to be able to interpret and use them.

7.6.3 Composed Modelling Options

Actual networks may need more than a single communication stack that can be arranged in several configurations: in particular, here we will analyse how two of the most popular configurations can be modelled according to the IoT Communication Model. In the following we will refer to (1) *gateway configuration* as the composition of two or more protocol stacks that are placed side by side across different media, and (2) *virtual configuration* as the composition of two or more protocol stacks, one on top of the other.

7.6.3.1 Gateway Configuration

In this configuration, the IoT Communication Model describes the overall communication behaviour of the system so that any two communicating element can be seen seamlessly connected.

Figure 7.18 provides a graphical example of the modelling of three protocol stacks in gateway configuration. In this example, the two end-point Application Layers can communicate thanks to the gateways which maps the underline stacks.

In particular, the first gateway (on the left of the figure) bridges the communication between an Ethernet and a WiFi network, while the second (on the right), in addition to the bridging functionality between WiFi and ZigBee, adds a translation functionality for converting IP to 6LoWPAN, TCP to UDP, HTTP to CoAP and vice versa.

This gateway configuration may be used in the recurring example to let the Mote Runner Node communicate using ZigBee technology with the Alarm Service deployed in a server farm thanks to the two gateways.

While the actual configuration of the different protocol stacks is out of the scope of the model, the overall behaviour of the system can be modelled according to the five interoperability aspects described above.

7.6.3.2 Virtual Configuration

In this configuration the IoT Communication Model aims at describing the overall communication behaviour of a system, where the actual communication path is virtualised by tunnelling the communication using a second protocol stack.

Figure 7.19 exemplifies the modelling of a system behaviour using a virtual configuration: here, there is an inner communication path composed of an Ethernet network and a WiFi network using a bridging block and an outer communication path that is independent of the inner path and allows for the two application layers to communicate. Such a scheme is usually realised using virtual private network solutions.

7.6.4 Channel Model for IoT Communication

This model aims at detailing and modelling the content of the channel in the Shannon-Weaver model in the context of the IoT domain. This model does not pretend to capture every possible characteristic of IoT technologies, but provides a common ground to be used to compute overall system performance and for benchmarking. Further models have to be considered in order to account for more specific physical aspects.

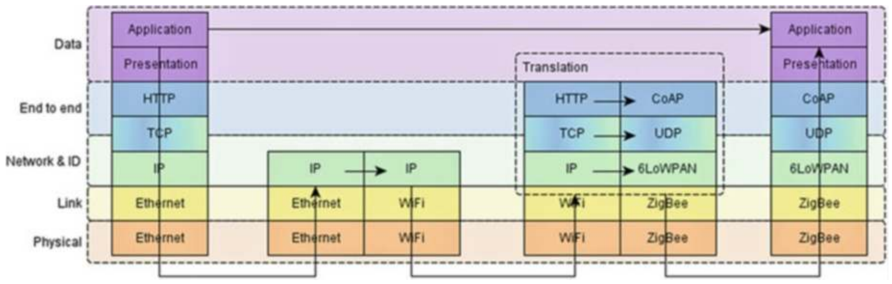


Fig. 7.18 Gateway configuration for multiple protocol stacks, aligned according to the IoT aspect model (see Fig. 7.17)

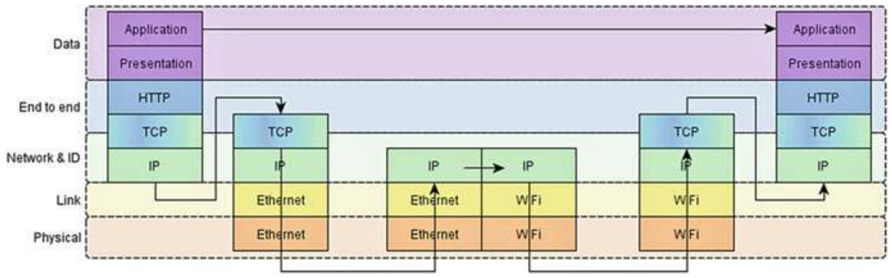


Fig. 7.19 Virtual configuration for multiple protocol stacks

Figure 7.20 depicts end-to-end abstraction of a packet delivery between distant Devices. The information source can be abstracted as a resource in the IoT Domain Model, and the transmitter as a Device; while the receiver and destination pair can be mapped as a Device-Service pair.

Following this abstraction, and pushing it forward, we focus on the channel modelling. In the IoT context, the channel can assume a multiplicity of forms.

Please notice that the following abstraction is useful in order to have an abstract description but when it comes to apply the Shannon-Hartley theorem it is crucial to remember this theorem has to be applied independently to each link composing the path between the sender and the receiver: $C_I = B_I \log(1 + S_I/N_I)$, where C_I is the channel capacity, B_I is the channel bandwidth, S_I/N_I is the signal-to-noise ratio (or the carrier-to-noise ratio in case of modulated signals), each of them related to the I-th link. This channel capacity metric is concave and it can be aggregated according the following rule: $C_{i,k} = \min(C_{i,j}, C_{j,k})$, where $C_{i,k}$ is the aggregated capacity from i to k, while $C_{i,j}$ is capacity of the link from i to j and $C_{j,k}$ is the capacity of the link between j and k.

Given two adjacent channels, which require to be connected by the means of a gateway, their aggregated capacity is extremely useful in order to dimension the gateway itself. Nonetheless, assuming you cannot control the routing on the Internet the scope is limited to the portion of links of which you know the characteristics, or for a link of which you can suppose to know the lower bound. A valid assumption

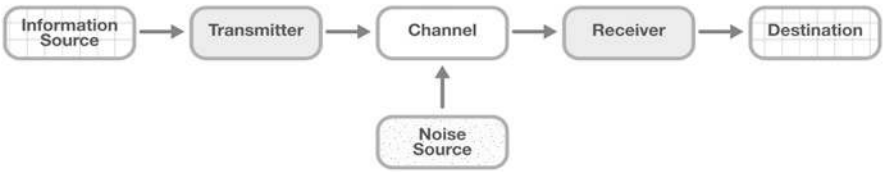


Fig. 7.20 Schematic diagram of a general communication system

will be anyway that the aggregated capacity cannot be bigger than the capacity of the known links, providing a strong tool to avoid over-dimensioning the gateways. Indeed, this is extremely useful when designing a constrained network and its ingress and its egress.

It is important to point out that there is a distinction between the channel model in the current Internet and that of the IoT. The former is depicted in Fig. 7.21 below, where the Internet block acts as a black-box summarising every channel transformation that may happen between the two gateways.

To proceed in modelling the channel in IoT it is important to give a definition of what we call constrained and unconstrained networks:

- **Unconstrained networks** are characterised by high-speed communication links (e.g., offering transfer rates in the Mbit/s range or higher) like, for instance, the wired Internet of today. Link-level transfer latencies are also small and mainly impacted by congestion events in the network rather than by the physical transmission technology;
- **Constrained networks** are characterised by relatively low transfer rates, typically smaller than 1 Mbit/s, as offered by, e.g., IEEE 802.15.4. These networks are also characterised by large latencies. These are due to several factors including: (1) the involved low-bitrate physical layer technology and (2) the power-saving policy of the terminals populating these networks, which may imply the periodic power off of their radios for energy-efficiency reasons.

According to this *heterogeneous networks* can be seen as the combination of *constrained* and *unconstrained networks* linked together via gateways and/or proxies.

The picture is much different in the IoT. As can be seen in the scenarios depicted in (Rossi 2013), in the simplest IoT case the channel consists of a single constrained network (e.g. a WSAN island), as depicted in Fig. 7.22.

In a slightly more complicated case, the IoT channel can consist of several constrained networks, which can rely on different network technologies (see Fig. 7.23).

A different case consists of a channel embodied by a constrained network and an unconstrained one (see Fig. 7.24).

An additional case consists of a channel formed by two constrained networks intermediated by an unconstrained network. A common implementation of this case us the most important in the IoT: the one involving two constrained networks linked by the Internet (see Fig. 7.25).

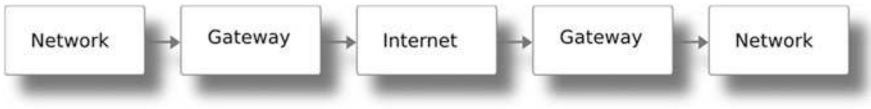


Fig. 7.21 Channel model for the current Internet

Fig. 7.22 IoT channel for a single constrained network

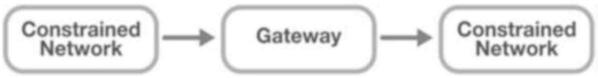


Fig. 7.23 IoT channel for communication over two constrained networks

What makes IoT very peculiar is the nature of the constrained networks it relies on. Such networks are formed by constrained Devices and the communication between the Devices can:

1. Be based on different protocols;
2. Require additional processing in the gateways.

It is important to point out that the characteristics of each network can have a noticeable impact on the overall end-to-end communication.

In the previous section we tackled the channel capacity using the Shannon-Hartley theorem and the min operation in order to aggregate multiple hops. Obviously the channel capacity is not the only important metric when modelling the IoT communication.

7.7 Trust, Security, Privacy

IoT systems integrate in a seamless way physical objects, data, and computing devices into a global network of information about ‘smart things’. In this scenario, services act as bridges through which these ‘smart things’ interact with each other in an automated way and with as less human intervention as possible. Towards our aim to provide a Reference Architecture for IoT systems, it becomes thus mandatory to discuss potential security issues and define a security model for our architecture. On the way to our goal we proceed as follows: we identify a few separate classes of security properties that we deem important for an IoT system and provide, for each class, tools and mechanisms that serve as solid foundations upon which we can build complex solutions that guarantee those properties.

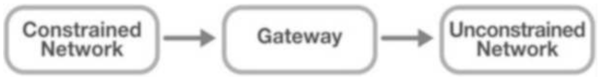


Fig. 7.24 IoT channel for communication constrained to unconstrained networks

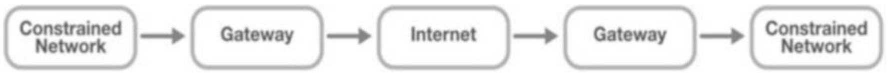


Fig. 7.25 IoT channel for communication over two constrained networks intermediated by the Internet

Considering the multi-faceted entities that a IoT system is made of, we spot the following necessary properties: Trust, Security, Privacy, and Reliability. In the remainder of this chapter we discuss these properties separately and delineate, for each of them, a reference model within the framework of our architecture.

7.7.1 Trust

An important aspect of IoT systems is the fact that they deal with sensitive information (e.g. patients' electronic health records). The entities and services therein recurrently process, store, retrieve, and take decisions upon this type of data. In this scenario, enforcing trust – compliance to an expected functional behaviour – on all entities, protocols, and mechanisms an IoT system is made of becomes a "must".

Within this project, we focus on Trust at application-level. In particular, we aim at defining a Trust Model that provides data integrity and confidentiality, and endpoint authentication and non-repudiation between any two system-entities that interact with each other.

Trust Model Mandatory Aspects

Describing all possible trust-model archetypes is out of the scope of this document. Nonetheless, we list hereafter a few and basic aspects that we believe to be mandatory for defining a Trust Model for IoT systems:

- The Trust-Model **domains**: In complex systems that include multi-faceted entities, like, e.g., the IoT, a model that equally shapes the Trust of all components is difficult, if not impossible, to define. For this reason, various domains within the system should be determined, with every domain defining a specific set of subjects to which certain aspects of the trust model apply;
- Trust-**evaluation mechanisms**: They define a coherent and safe methodology for computing the trustworthiness degree of a subject within the system. Evaluation mechanisms are based on information previously collected on the given subject. Depending on the application scenario, this information can be obtained

by direct experiences with the subject, witness information on the subject coming from other members of a community, social-network analysis providing sociological information on the subject and so on. A trust-evaluation mechanism must take into account the source of the information on which the trust value is being computed, i.e. the trustworthiness of the source itself, and carefully weight its information accordingly in computing the final trust value;

- **Behaviour policies**: They regulate the ways two subjects within the same Trust Model domain interact according to their trustworthiness value. They define how subjects that use the system may interact with other subject. E.g., if a wireless sensor A is asked to handle a multi-hop message coming from a sensor B with a very low trust value, Sensor A might decide, according to the behaviour policies defined by the Trust Model, to not accept the message from Sensor B. Though it is not recommended, a Trust Model can define specific behaviours for interacting with subjects whose trust-value cannot be computed within that model;
- **Trust anchor**: It is a subject trusted by default (possibly after authentication) by all subjects using a given Trust Model, and exploited in the evaluation of third parties' trustworthiness. In the IoT environment the trust anchor can either be local to a given subnetwork – running on a node in the same peripheral network, e.g. a gateway – or a global and centralised device that is deployed on the Internet;
- **Federation of trust**: It delineates the rules under which trust relationships among systems with different Trust Models can be defined. The federation of trust is essential in order to provide interoperability between subjects that use different Trust Models. The federation of trust becomes particularly important within an IoT system deployed on a large scale, where the coexistence of many different Trust Models it is very likely;
- **M2M support**: The interaction among autonomous machines is deemed very common in IoT systems. Prior dynamically identifying and accessing resources of one-another, these machines should be able to autonomously, according to the specifics in the Trust Model, evaluate the trustworthiness of each-other.

7.7.2 Security

Now that we have discussed the fundamental aspects that will be included in our Trust Model, in this section we provide a generic overview of the Security reference model in our architecture.

Our Security reference model is made of three layers: the Service Security layer, the Communication Security layer and the Application Security layer. The former, described in details in (Gruschka and Gessner 2012), includes the following components: Authorization, Identity Management, Trust and Reputation, Authentication, and key exchange and management. In the remaining of this section we detail the two last layers.

7.7.2.1 Communication Security

IoT systems are heterogeneous. Not only because of the variety of the entities involved (data, machines, sensors, RFID, and so on), but also because they include Devices with various capabilities in terms of communication and processing. Therefore, a Communication Security Model must not only consider the heterogeneity of the system, but it also should guarantee a balance between security features, bandwidth, power supply and processing capabilities (Rossi 2012).

Here we work under the assumption that the IoT device space can be divided into two main categories: constrained networks (NTU) and unconstrained networks (NTC) (see Networks and communication entities, Chap. 2 in (Rossi 2012)). The domain of constrained devices contains a great heterogeneity of communication technologies (and related security solutions) and this poses a great problem in designing a model encompassing all of them. Examples for such communication technologies can be found in (Rossi 2012)).

To mitigate the aforementioned heterogeneities we could provide a Communication Security Model with a high degree of abstraction. However, it would be useless, as it would lack the specifics needed in the moment of implementing a specific IoT architecture. As in the Communication Model (see Sect. 7.1.6), we address the problem by introducing profiles that group heterogeneous Devices into groups characterised by given specifications.

Figure 7.26 above depicts our approach to lower-layer security in IoT. We exploit gateways:

On the edge between the domains of unconstrained and constrained devices, gateways have the role of adapting communication between the two domains. Gateways are unconstrained devices; therefore, they can be exploited to boost up the security of constrained devices by running on their behalf energy-hungry and complex security mechanisms. In addition, gateways can also be used in order manage security settings in peripheral NTC networks.

We enable these functionalities in the gateways by extending them with the following features:

- Protocol adaptation between different networks (by definition);
- Tunnelling between themselves and other nodes of the NTU domain. (Optional; impacts on trust assessment);
- Management of security features belonging to the peripheral network (Optional);
- Description of security options related to traffic originated from a node attached to the gateway. (Authentication of source node, cryptographic strength, . . .);
- Filtering of incoming traffic (i.e. traffic sent to one of the nodes attached to the gateway or vice-versa) according to network policies, user-defined policies, and destination-node preferences (Optional).

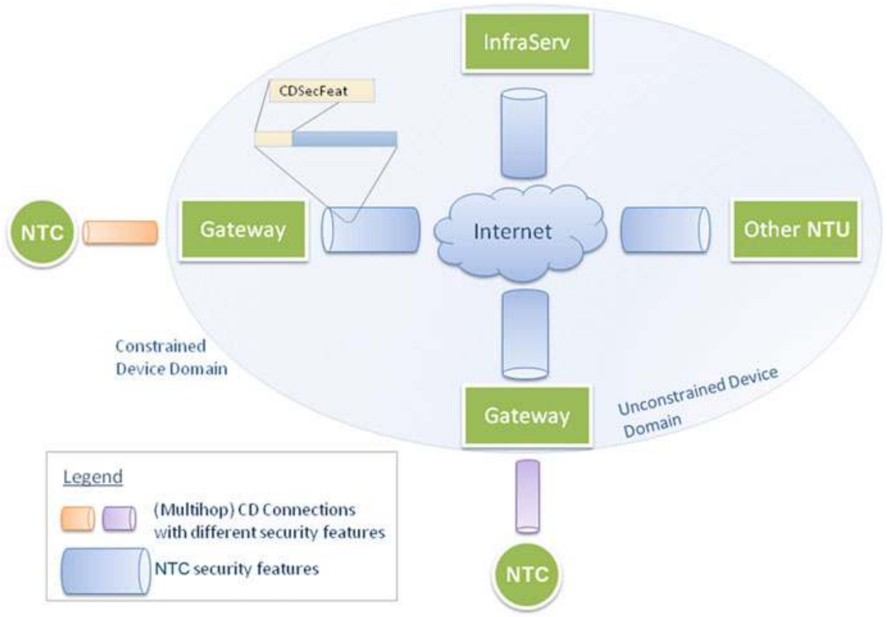

InfraServ
CDSecFeat
NTC
Gateway
Internet
Other NTU
Constrained Device Domain
Gateway
Unconstrained Device Domain
Legend
(Multihop) CD Connections with different security features
NTC security features
NTC

Fig. 7.26 *NTC* Constrained Device Network, *NTU* Unconstrained Device Network, *CDSecFeat* Constrained device security feature. The CDSecFeat implementation leverages the extension of the functionalities of gateway devices

7.7.2.2 Application Security: System Safety and Reliability

IoT systems include, without any doubt, a wide range of application scenarios: from home-security to structure monitoring of bridges, buildings, and so on, and from surveillance systems to health monitoring. Most of these scenarios must be reliable: a single failure in the system can lead to tragic consequences. This is why, besides from security and privacy mechanisms that guarantee trustworthiness of the system as a whole, it becomes important to assure also system safety.

System safety is application specific: for an electricity system safety includes assuring that no harm is done in case of a short circuit. For an elevator system safety would include making sure that the elevator does not start moving when the elevator doors are opened. Nonetheless, there is a common approach to achieve fail-safe systems made of two phases. The first, called the hazard identification phase, aims at detecting all possible risks that could possibly lead to severe accidents. The second phase includes the system design according to the fail-safe philosophy: systems are designed in a way that the far majority of failures will simply result in a temporary or total loss of service, so to avoid major damage/ accidents. An example of a safe-fail system is the security belt sensor in smart-cars: If the driver does not fasten it, the car does not start.

While we believe that the classical fail-safe approach to system design can assure safety in IoT systems, with respect to hazards inside the system (e.g. the security belt within the car, the short circuit within the electricity system and so on), we also believe that often, the safety of the system depends on issues that originate outside the system. The following scenario gives a representative example of outside-the-system hazards: A bulldozer aiming at bringing down a tree damages (by chance) the foundations of a building nearby. Even though the damage is not visibly spottable right away, at the first slight earthquake it makes the building crumble down by thus costing human lives.

Clearly, in these cases, threat analysis plays an important role. Despite from considering only system-insider hazards, the system designer should carefully examine the 'outside world' of the system in order to identify potential outside hazards. Only after a meticulous analysis of all possible threats (both insiders and outsiders) proceed with the system design following the fail-safe philosophy.

Lastly, another group of vicious threats imposed to safety, or rather, reliability of IoT systems are terroristic. These can either aim at bringing down large automatic systems e.g. a city or country wide electricity system, internet connectivity, border security monitoring system and so on, or targeting directly the users (e.g. by wirelessly reprogramming pacemakers of patients[2]). In the former case, the attack consequences could be limited by including intrusion/failure detection mechanisms (e.g. heart-beat protocols) coupled with redundancy that brings the targeted service up in a short-time period after the attack. In the second case, however, this type of solution might not work well: If the pacemaker of a patient is stopped, even though an alarm might be raised in the IoT system, the patient's life would most probably end in a short time.

7.7.3 Privacy

Due to the variety of the entities that handle user-generated data in IoT, guaranteeing data privacy becomes mandatory in these systems. For this reason we include in our reference model also a Privacy Model, the aim of which is to describe the mechanisms – e.g. access policies, encryption/decryption algorithms, security mechanisms based on credentials, and so on – that prevent data of a subject (either user or entity) to be used improperly.

According to (Weber and Weber 2010), a privacy friendly system should guarantee the following properties:

- The subject must be able to choose sharing or not sharing information with someone else;
- The subject must be able to fully control the mechanism used to ensure their privacy;

[2] According to a report published at www.secure-medicine.org, peacemakers can be wirelessly hacked in, and reprogrammed to shut down or to deliver jolts of electricity that would potentially be fatal to patients.

- The subject shall be able to decide for which purpose the information will be used;
- The subject shall be informed whenever information is used and by whom;
- During interactions between a subject and an IoT system, only strictly needed information shall be disclosed about the subject, and pseudonyms, secondary identity, or assertions (certified properties of the end-user) shall be used whenever possible;
- It shall not be possible to infer the subject's identity by aggregating/reasoning over information available at various sources;
- Information gained for a specific purpose shall not be used for another purpose. E.g., the bank issuing a credit card should not use a given client's purchase information (logged so to keep track of that client's account) to send him advertising on goods similar to his purchases.

To provide the above properties the IoT-A privacy model relies on the following functional components: Authentication FC, Trust and Reputation FC.

Table 7.1 below briefly summarizes how these components mitigate some of the privacy threats to privacy, further discussed in the threat analysis performed in IoT-A (see Appendix).

Central to the Privacy Model is the Identity Management Functional Component. A description of this FC is provided in deliverable (Gruschka and Gessner 2012).

In our system, any subject (service or user) is univocally mapped to its root identity. However, a subject might require to be provided with multiple secondary identities by the Identity Manager. The set of multiple identities associated to a unique subject is denoted with identity pool (see Fig. 7.27). Secondary identities can then be used, for privacy or usability purposes, when the subject interacts with the IoT system. However, the system does log the identities (either secondary/pseudo or root identities) of the subjects it interacts with so to mitigate possible Repudiation. The Identity Management FC provides a mapping functionality that maps (to requesters with the required credentials) root identities to secondary identities/pseudonyms.

The second corner-stone functionality for ensuring privacy is *Authentication* (AuthN component).

Its functionality is to bond a subject to its identity (root identity) or to certify properties/roles of the subject, or both. If the subject is a user, examples of possible certified properties can be:

- Has age over 18 years old;
- Has valid driving license;
- Has certification level x.

Similarly, certified roles can be:

- Management;
- Operational;
- Maintenance,...

Table 7.1 Example of privacy threats mitigation within IoT-A

Threat	Result	Mitigation
Identity spoofing	User's identity is spoofed	Robust user authentication procedure preventing man-in-the-middle attacks, with proper credentials-management policy provided by an **Authentication FC**
	User is involved in transactions with a malicious peer	Trustworthy discovery / resolution / lookup system. Trustworthiness of the entire system is guaranteed through its security components (especially **Authentication FC** and **Trust and Reputation FC**) as well as its global robustness (security by design)
Information disclosure	Attacker gains knowledge of user's private parameters	The **Identity Management FC** enforces a robust pseudonymity scheme that ensures anonymity and unlinkability
	Attacker gains knowledge of user's location	User's location can be hidden through reliance on pseudonyms provided by **Identity Management FC**

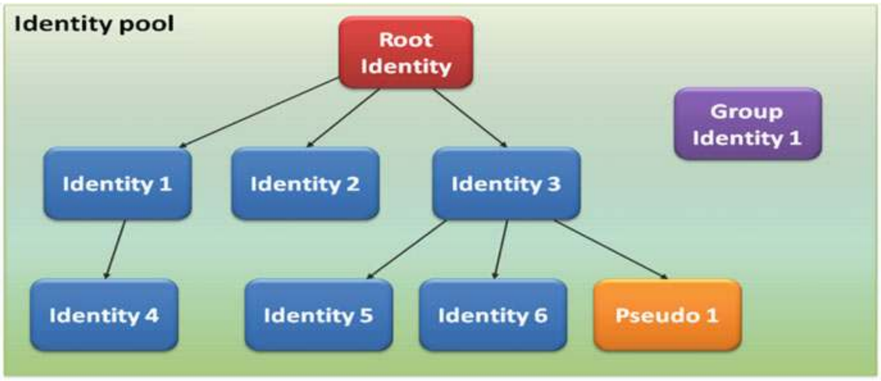


Fig. 7.27 Example of an identity pool

So, in our system, a given subject can be granted access to an IoT resource according to the subject's identification, or according to the subject's certified properties/roles. This enables subjects to still get access to the system yet not revealing their identity.

The AuthN component proposed by IoT-A offers the Authenticate functionality, the profile of which is:

assertion: Authenticate (UserCredential)
where UserCredential is any kind of information used by the Authenticate functionality to check the identity of the party to be authenticated (e.g. username – password pair, PIN code, retinal identification and so on).

assertion (following definition of (Gruschka and Gessner 2012)) is the information that guaranties the occurrence of an authentication of a user client at a particular time using a particular method of authentication. The assertion is further used by the Authorisation (AuthS) component in order to decide upon granting or denying access to a resource.

Finally, the AuthN component provides also *Authorisation* (AuthS): It is the process by which access to information or an IoT Resource is granted to a subject, according to an access policy and for a specific type of action. In order to guarantee user-privacy, the end-users should be in control of access policies relating to their personal data.

The profile of the Authorise function is:

Boolean: Authorise (Assertion, Resource, ActionType),
where Assertion is the result of Authentication, Resource represents the resource to be accessed, and ActionType represents the action to be performed upon the resource.

As mentioned earlier, there are various models of authorisation, property-based access control and assertion-based access control (Gruschka and Gessner 2012). Both are supported by IoT-A through abstract APIs (Gruschka and Gessner 2012).

Identity Management FC, Authentication FC, and Authorisation FC guarantee privacy within the IoT system. Nonetheless, if the data within the IoT system's database is stored as clear text, nothing prevents hackers from tampering with the database and accessing the data. To protect the user against these types of attacks, we believe that the data should be encrypted before storing it in the database.

7.7.4 Contradictory Aspects in IoT-A Security

In distributed systems, including IoT-like ones, one has often to trade-off between security properties. In particular, trust and privacy, are considered as being two contradictory properties. From one side, we want to build a system which is trustworthy. I.e., every entity in that system can prove, according to either trust-building mechanisms or to certificates distributed by some authority, its own trust value. From the other side, we want the system to provide, to each entity, the privacy that it requires, without forcing it to disclosure more personal information that it wants to. This tension between security and privacy emerges also in our reference model. Indeed, the trust-evaluation mechanisms for example not couple well with the many pseudonyms an entity might present to protect its privacy in various scenarios. Indeed, a given malicious entity can fool the system by presenting, within a given context, the pseudonym with the highest trust value built so far. It becomes thus very important to strongly bind, somehow, the trust value of an entity with its root ID. But, from the other side, this imposes problems to the privacy of the entities: If the trust-value has to be calculated on the fly, based on certificates given to that entity in the many interactions it has had in the past, all bound to its root ID, the entity can be easily traced inside the IoT, even though it presents different pseudonyms. A solution to this problem is to make the trust value be recalculated, each time an interaction occurs, by a unique, trustworthy system component which is also able to bind various pseudonyms to root IDs. This solution does guarantee correct trust values for all entities in the system, yet preserving their privacy. However, it has two major drawbacks: (1) The unique component would become a huge bottleneck in the system; (2) It would become a single point of failure: By compromising it (or tampering with it) an attacker would be able to de-anonymize all entities in the system, or even change trust-values to his liking, by boosting trust-values of malicious entities, and lowering the trust value of others.

For the above reasons, we believe that within the IoT-A system we should opt for a mechanism which trades-off trust for privacy: Subjects are allowed just one trust-value, valid for a certain number of pseudo-identities, and included in a trust-certificate signed by the AuthN component. The trust value is then updated each time the subject interacts in the system, by the counter-part of this interaction. The trust value is to be used for sensitive interactions and/or access to sensitive system resources, data, and services, within which the subject is thus required to present one of the pseudonyms bound to the trust-certificate. This way, a subject cannot fake its unique trust value, which is, from the other side not bound to its pseudonyms "trust-free" – the ones through which the subject can access less system's resources, data, and services, that do not require proof of trust values.

7.8 Conclusion

In this Section we introduced the foundation of the IoT ARM, the IoT Reference Model. The IoT Reference Model defines the basic concepts, models, terminology, and relationships in the IoT ARM. It demonstrates our thinking, rationale and design space for structuring the domain of the Internet of Things. It also proposes the Functional Groups that we deem relevant for IoT architectures, as outlined in the IoT Functional Model (see Sect. 7.1.5).

Within the IoT Reference Model, the IoT Domain Model was discussed in great detail, as the IoT Domain Model defines the language, the concepts, and the entities of the IoT world and how they are related to each other. This is confirmed by the fact, as we learn in Section (sec: Chap. 6 "IoT Context View"), that the IoT Domain Model plays a prominent role in IoT-A-guided system architecting. As we will see in Chap. 12 when we perform a reverse mapping analysis with the concepts defined in other projects and standards related to the Internet of Things, the definition of a common understanding is crucial for developing interoperable architectures and systems. This common understanding permeates every aspect of the architecture, and will be a key aspect for the widespread acceptance of a future IoT systems and standards. In that respect it is most important to carefully study the concepts of the

IoT Domain Model, as it is the foundation of the other models presented in this chapter and of the IoT Reference Architecture that will be discussed in Chap. 8.

While a common language and common terminology is the precondition for all other models, this chapter also provided the other models crucial for the development of IoT architectures, most importantly the IoT Information Model that relates to important aspects of information in an IoT system and will be detailed in the IoT Information View in Sect. 8.2.3 that discusses information on a higher level of detail.

The IoT Functional Model discussed in this chapter defines several Functional Groups that pick up the IoT concepts and entities introduced in the IoT Domain Model and it relates them to common functionalities present in an IoT architecture. Just as for the IoT Information Model and View, the IoT Functional Model will be further detailed with concrete functional components in Sect. 8.2.2.

Finally, Communication and Security models, as well as techniques of system safety and reliability where introduced that address these issues in IoT. The security and the communication model constitute Functionality Groups in the IoT Functional Model, and will be picked up again in the IoT Reference Architecture (see Sects. 8.2.4 and 8.3.3).

What we have also addressed in this chapter, is the application of the common IoT use case introduced in Sect. 4.2 to several models in order to facilitate getting acquainted with the concepts defined in the respective model by tying their understanding together with a common, "Red Thread". We hope that this application of the use case helps with understanding the different models. We are aware of the complexity of the IoT Domain Model and the Trust, Security, and Privacy issues, but this complexity is inherent in the domain of the Internet of things itself. It is however crucial to really understand the models introduced in this chapter, before moving on.

The next Chap. 8, the IoT Reference Architecture, builds upon this foundation and details it even further, so that concrete IoT-compliant architectures can be derived. The section uses several ways of projecting the IoT Reference Architecture, and it also presents several "Views" that complement the different models presented in this section. For instance, we propose Functional Components, which relate to the IoT Functional Model and the IoT Communication Model, in the Functional View (see Sect. 8.2.2) that we discussed in this chapter. We also provide an Information View, which tightly relates to the IoT Information Model discussed in this chapter.