

Chapter 6

A Process for Generating Concrete Architectures

Mathieu Boussard, Stefan Meissner, Andreas Nettsträter, Alexis Olivereau, Alexander Salinas Segura, Matthias Thoma, and Joachim W. Walewski

This chapter addresses the question of how to generate concrete architectures with the IoT ARM, which is one of the many uses to which an architectural reference model can be put (see Chaps. 3 and 4). This topic was already touched upon in Section “Generation of Architectures” in Chap. 3, but it is covered in greater depth in this section.

Note that we do not prescribe any specific architecting methodology for generating concrete architectures. Instead, this section outlines how and where

M. Boussard (✉)

Alcatel-Lucent Bell Labs France, Route de Villejust, 91620 Nozay, France
e-mail: mathieu.boussard@alcatel-lucent.com; www.alcatel-lucent.com

S. Meissner

University of Surrey, Stag Hill, Guildford GU2 7XH, UK
e-mail: s.meissner@surrey.ac.uk; www.surrey.ac.uk

A. Nettsträter

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

A. Olivereau

Commissariat à l’Energie Atomique, Avenue des Martyrs 17, 38054 Grenoble, France
e-mail: alexis.olivereau@cea.fr; www.cea.fr

A. Salinas Segura

University of Würzburg, Josef-Stangl-Platz 2, 97070 Würzburg, Germany
e-mail: alexander.salinas@uni-wuerzburg.de; www.bwl.uni-wuerzburg.de

M. Thoma

SAP AG, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany
e-mail: Matthias.thoma@sap.com; www.sap.com

J.W. Walewski

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

during the architecting process the IoT ARM can provide help and input for the architect. We return to this topic of “methodology agnosticism” in Sect. 6.2.

As can be seen in Chap.3 Figure “Process for the generation of concrete architectures”, the IoT ARM informs the engineering strategies for the design of a concrete IoT system, and the transformation rules are derived from the entirety of the IoT ARM. Also, the IoT ARM informs the requirement-generation process. In this section we are focusing in greater detail on the generation of requirements and on the transformation of these requirements into a concrete architecture. Notice that a concrete architecture implies that it meets a selected use case and application scenario.

6.1 Process Steps to Generate IoT Architectures

What are the main building blocks of a domain-specific architecture that adheres to the IoT ARM framework? The answer is: architectural views. As discussed at the beginning of Chap. 7 and Sect. 8.1 “Short Definition of Views and Perspectives” in Chap. 8, we chose to arrange a system architecture according to views, with the totality of all views constituting the architecture description. Figure 6.1 outlines how the views are related to each other and how they contribute to the system design. All views shaded yellow are covered in detail in the IoT Reference Architecture (see Chap. 8) or in this Section. These views are:

- Physical Entity view
- Deployment view
- Operational view
- IoT Context view
 - IoT Domain Model
- Functional view
- Information view

In this figure:

- In dark red: views that are treated in Chap. 8 and in Carrez et al. (2013) or in this section;
- In orange: related models (see Chap. 7).

Note that since the IoT Domain Model also encompasses the role of users, it actually implicitly covers the enterprise view as advocated by RM-ODP (Raymond 1995) (see Chap. 3 for a discussion of the enterprise view and Sect. 7.3 for a discussion of roles in the IoT Domain Model). Note also that although the other views shown in Fig. 6.1 (“... view”) are not covered in the IoT ARM, this does not imply that they are not important for generating concrete architectures. This

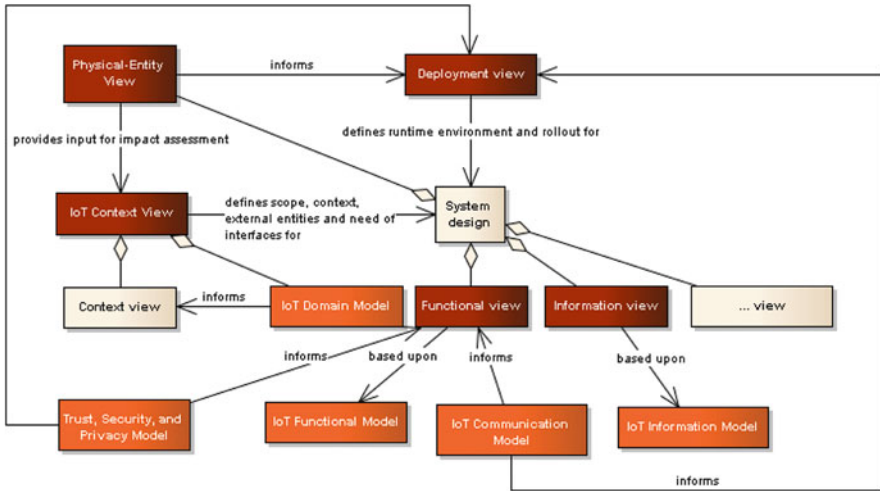


Fig. 6.1 Relationship of architectural views (based on Fig. 15-1 in Rozanski and Woods 2011)

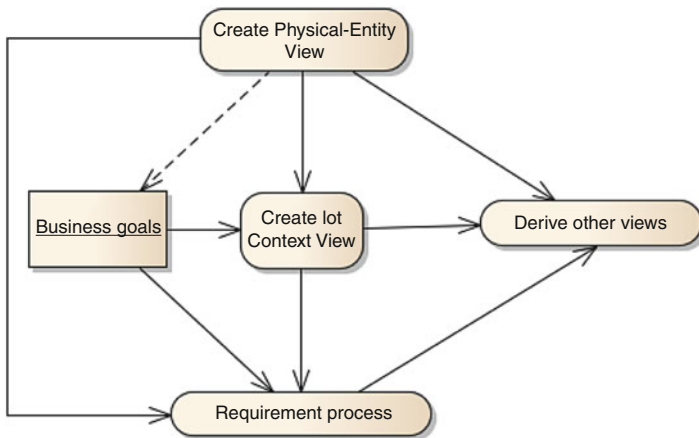


Fig. 6.2 UML activity diagram of the IoT architecture generation process (generation of requirements and transformation of requirements into a concrete architecture)

becomes clearer when we look at the architecture generation process in more detail. Figure 6.2 outlines the activities involved in generating an architecture. These are:

- Create Physical Entity view
- Create IoT Context view
- Requirements process
- Derive other views

In this figure, *dashed arrows* represent dependency, while *solid arrows* represent control flow (can be understood as either the next step or expressing a logical contingency of the target on the source).

As you can see, the creation of the Physical Entity View and IoT Context View (see Fig. 6.1) are explicit activities in the architecting process. All other views are comprised in the activity “derive other views”. Before we look at each of these activities in more depth, let us return to the question of architecture methodologies and how the IoT ARM relates to them.

6.2 Compatibility with Other Architecting Methodologies

Figure 6.2 could give rise to the impression that we prescribe a sequential approach for generating architectures: (1) Define the scope, i.e. the business goals; (2) Create the Physical Entity View and the IoT Context View; (3) Define requirements; and (4) Generate the remaining views. This type of sequential approach to architecting lies, for instance, at the heart of the waterfall approach (Royce 1970). This interpretation of Fig. 6.2 is indeed true if all arrows in Fig. 6.2 are understood as arrows in time. However, they can also be understood as logical dependencies. For instance, in order to conduct the requirements process, we need a set of formulated business goals, an IoT Context View and a Physical Entity View. If we interpret the process described in this Section in the latter way, it can be mapped onto a plethora of popular architecting methodologies, such as Model-Driven Engineering (MDE) (Miller and Mukerji 2003), Pattern-Based Design (Gamma et al. 1994), and the Spiral Model (Boehm 1988).

The only limitation we see is in the choice of views. Some architectural methodologies prescribe different sets of views. Some of them, for instance the 4+1 approach, lack some of the views we prescribe (mainly the information and context views) (Kruchten 1995). In this case we could choose to embed the 4+1 framework into the process described in this Section. On the other hand, other methodologies comprise views that are not part of the IoT ARM set. In this case, the option is to integrate the IoT ARM views (and the manner in which they are derived) into this other methodology.

6.3 IoT Architecture Generation and Related Activities

Since neither the IoT Context View nor the Physical Entity View are addressed in the IoT Reference Architecture (see Chap. 8), and since they are integral parts of the architecting process (Fig. 6.2), we need to look more closely at both of these views and understand how they inform the architecting process.

6.3.1 *Physical Entity View*

Before we describe the Physical Entity View we need to discuss what it is not: i.e. the “traditional” physical view in system architecting – a well-established view in software system architectures (see, for instance, Kruchten 1995). “It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is also known as the deployment view.” (Wikipedia 2013a; 4+1 view). As Fig. 6.1 implies, we are not using the term physical view for the deployment view in order to avoid semantic tension with the Physical Entity View.

The Physical Entity View does of course refer to the Physical Entity in the IoT Domain Model (see Sects. 7.3.2 and 9.1). The Physical Entity is “any physical object that is relevant from a user or application perspective” (Appendix). For a concrete use case and application scenario, this is of course a well-defined set of physical objects. For instance, in the recurring example (see Sect. 4.2), the Physical Entities are the orchids that are transported in a truck and these orchids are subject to environmental monitoring.

It is obvious for many reasons why the architecture of an IoT system also needs to include a Physical Entity View. Firstly, the dimensions, the distribution and the properties of the Physical Entities have various implications. Examples of these implications are:

- **Devices:** the sensors/actuators needed and where are they situated; their relationship to the Physical Entity (directly mounted; touching; remote but in sight . . .), etc. Note that the device choice is influenced by the Physical Entity. In the recurring example, it is too expensive (in relation to the market price of the Physical Entity) to measure the temperature of each orchid. Instead, sensors that measure the air temperature are situated inside the cargo area. It is then assumed that the air temperature equals that of the orchids. In other words, the Physical Entity model also needs to include a sensing and/or an actuating model.
- **Information view:** what physical quantities are monitored by the sensors; how are the quantities related to each other, etc.? In the recurring example the quantity that is handled by the system is the air temperature in the cargo area of the truck.

Secondly, in some use cases, the devices might be incorporated inside the Physical Entity, which can have a range of implications for the IoT system. For instance, if sensors are deployed inside a human body and the wireless sensor signal is to be relayed to an outside reader, we need to understand the in-body propagation characteristics of this signal. It may be the case that the strong attenuation caused by the body tissue calls for a scenario in which signal repeaters are deployed. This has implications for the communication aspect of the architecture (→ functional view).

Thirdly, the type of the Physical Entity – in combination with the application scenario – can have implications for the Trust, Security, and Privacy Perspective (see Sect. 8.2.3). Let us look again at the recurring example. Since orchids can be

very expensive, and since this can increase the likelihood of the truck being raided while, for instance, parked during a coffee break or overnight, it is paramount that the wireless signal emanated by the orchid monitoring system cannot be identified as such nor be deciphered. If this were to happen, it could, for instance, inform a burglar how many orchids are in the shipment.

Although the Physical Entity View is obviously very central to the IoT ARM, it is not covered in the IoT Reference Architecture. This apparent contradiction is attributed to the overwhelming range of Physical Entities in the IoT: they can range from the nano- and micrometre scale to truly macroscopic dimensions (e.g. glacier monitoring); they can be gaseous or liquid; they can be animate or inanimate or a mixture of both; they can be stationary or mobile. “Mobile” can include walking, running, moving on wheels, flying, coasting under water, flying through interplanetary space, and so on. Also, there is no ONE physical quantity to be monitored – in one use case it can be the temperature of orchids, in another the occupancy of a room (automated light switch), in another case blood sugar levels. This overwhelming range of Physical Entities provides for the generation of generic yet comprehensive viewpoints and thus models for the Physical Entity View. This lack of “least common denominator” is the reason why it was not possible to devise Physical Entity models at the reference architecture level and thus integrate them into the IoT ARM.

The user of the IoT ARM is advised to use his own domain understanding to devise the Physical Entity view. Where required, pertinent models (for instance, freshness vs. room temperature model for orchids) either need to be developed by the architecture team or they can be extracted from outside sources (literature, standards, etc.).

6.3.2 IoT Context View

As indicated in Fig. 6.1, the IoT Context View consists of two parts: the context view and the IoT Domain Model. The context view is an architecture view that is 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). To be more specific, the context view describes “what the system does and does not do; where the boundaries are between it and the outside world; and how the system interacts with other systems, organizations, and people across these boundaries” (Rozanski 2013). The concerns addressed by the context view are (Rozanski 2013):

- “System scope and responsibilities
- Identity of external entities and services and data used
- Nature and characteristics of external entities
- Identity and responsibilities of external interfaces

- Nature and characteristics of external interfaces
- Other external interdependencies
- Impact of the system on its environment
- Overall completeness, consistency and coherence”

Note that at least one of the concerns, i.e. “impact of the system on its environment” also applies to the Physical Entity, and investigating this concern thus requires input from the Physical Entity View (see Fig. 6.2).

A detailed example of a context view, including a context diagram and a description of the system components, can be found in Chap. 11.

Note that the context view focuses mainly on what lies outside the system and how the system interfaces to the outside world. This is sufficient for “generic” architecting processes, but for the IoT domain, not only do we know more about the system to be devised, we should actually also gather more information about the system at a very early stage in the architecting process. Why? Firstly, since IoT systems have many aspects in common by virtue of operation in the same domain, a lot of concepts are recurring concepts. One of the goals of the IoT ARM is to avoid “reinventing the wheel”, namely to avoid discovering, analysing and naming the very same aspects every time an architecture is generated. In order to permeate the entire architecture description with this understanding, we prescribe its use early on in the architecting process. This has advantages not only for the architecture generation itself, but also for other usages, such as architecture reuse. If common concepts, semantics, structures and relationships are fused into the core of an architecture description, this makes it much easier to reuse aspects of the architecture description or even the entire architecture. This can, for instance, be interesting for architecture development within a technology roadmap. Also, trust, security, privacy and safety are contingent upon system borders and thus on the functionalities and hardware that reside inside and outside the system border. The IoT Domain Model readily comprises both the “inside” and the “outside” of a system, and thus provides a deeper insight into relationships between the system entities and also interactions with the “outside world”. For all of these reasons, it is beneficial to conduct a domain model analysis before embarking on actions such as threat analysis and requirements engineering.

So what other reasons are there for expanding the context view “inward”, namely also covering the system itself? Why not just add a view to the architecting, namely the IoT Domain View, to the architecture description? The main reason is that both models are complementary and need to be applied early on in the architecting process. This is why we chose to pair the two system views. Note that the context in the IoT Context View has an extended meaning to that in the “traditional” context view, where it alludes to the context in which the system finds itself in relation to its surroundings. The IoT Context View expands on this by also including the entities within the system and by setting each of these entities in relation – context! – to the other entities.

The IoT Domain Model, on the other hand, provides a semantic and ontological overlay for the context view in that it provides guidance on which entities make up an IoT system and how they relate to each other. It also helps to identify system boundaries, which is one of the main questions to be addressed in the context view.

For more information on the IoT Domain Model see Sect. 7.3, and for guidance on how to generate a concrete IoT Domain Model see Sect. 9.1.

Note, that since all are listed and characterised in the IoT Context View, this is also the natural place for where to address the roles of all entities. These roles can for instance, be categorised as permissions, prohibitions, and obligations. For more information on these categories the reader is referred to elsewhere in the literature (Raymond 1995). For a discussion of how these roles figure into the system composition see Sect. 7.5.2.1).

An exhaustive discussion of the context view is available in literature (Woods and Nick 2008), but in order to enable immediate usability of the IoT ARM, we provide a short summary below.

6.4 Requirements Process and “Other Views”

6.4.1 Requirements Process

So far, we have shed light on two of the views that constitute an IoT architecture: Physical Entity View and the IoT Context View. Now we will discuss the remaining mandatory activities for generating an architecture: the requirements process and the derivation of “other views” (see Fig. 6.2). Figure 6.3 illustrates the architecture activities in more detail. How exactly the IoT ARM contributes to each of these actions is covered in the next Section.

As indicated in Fig. 6.1 and discussed in the previous Section, the context view is expanded by the IoT Domain Model. Therefore, both the generation of the “traditional” context view (see Sect. 4.1) and the expansion of this view in the IoT Domain Model are included in the creation of an IoT context view. As also explained in Sect. 6.4.1, the Physical Entity View provides input for the generation of the IoT context view.

With the input from the Physical Entity View and the IoT Context View, we can conduct a threat analysis. This type of analysis identifies potential weaknesses of the system use case envisaged, and it also identifies design choices and in some cases even functionalities that mitigate the risks identified. This analysis also provides guidance for the requirements engineering action (the security risks that need to be addressed by requirements).

The requirements process consists of many intermediate steps. The requirements engineering action generates a list of references that belong to one of three types: view requirements (i.e. requirements that directly inform one of the architectural views), qualitative requirements and design constraints. Note that we categorise the Unified Requirements (see online at <http://www.iot-a.eu/public/requirements>) along different dimensions (functional requirement, non-functional requirement, . . .) in order to increase the usability of UNIs for users who are not familiar with the

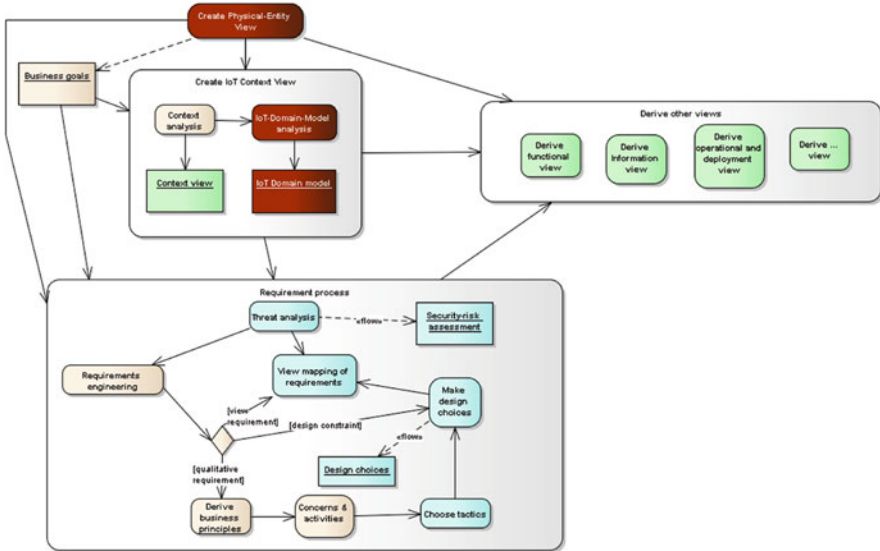


Fig. 6.3 IoT architecture generation (expansion of Fig. 6.2)

IoT ARM taxonomy of requirements. The translation of the UNI requirement types into IoT ARM process types is described Sect. 6.7.

In the prevailing approaches toward translating qualitative requirements into view-related requirements, we usually rely on a set of view requirements that is already available. An example of this type of approach is Quality-Function Deployment (Erder and Pureur 2003), which, amongst other things, is a central part of the ISO 9000 standards suite (ISO 2009). The assumption of an existing set of view requirements is a reasonable one for straight-forward product extensions or the design of simple systems, but for most IoT systems, this type of approach is not feasible. In other words, qualitative requirements cannot be translated directly into view requirements. In typical IoT systems, not only is complexity high, but there is often a plethora of options for achieving the desired performance of the system to be built. In other words, there are many sets of view requirements that meet the same set of qualitative requirements.

In order to overcome this design roadblock, we devised a step-by-step process through which view requirements can be inferred from qualitative requirements. The first step is to formulate the rationale of the qualitative requirements as business principles. For a detailed discussion of business principles, see (Rozanski and Woods 2011). This step is followed by identification of concerns and related activities. This action includes identifying each of the qualitative requirements with one or more architectural perspectives. The next step is to choose design tactics and then make design choices (covered in more detail in Sect. 6.4.1.8).

If the requirement is a design constraint, then it directly informs the design choice action.

From the design choices made, it is then possible to formulate implications for the functional view and other views (see Sect. 6.4.1.8).

If the requirement is of the view type, it can later be mapped directly onto the architecture description. We have found it very helpful to initially map functional view requirements onto the functional decomposition (see Sect. 8.1) throughout the requirements process. This makes it easier to track what parts of the system architecture are already covered by the requirements and whether more requirements are needed.

Salient inputs to the requirements process come, of course, from the Physical Entity View. This view, among others, provides the requirement engineer with information about special features of the “things” and the device-thing relationship (see Sect. 6.4). Another important source of information is the IoT Context View. It not only provides an overview of the system envisaged, but, thanks to the IoT Domain Model, it also provides the requirement engineer with information about the entities that are part of the system, what they are called, and how they relate to and interact with each other.

6.4.2 *View Derivation*

The remaining views are addressed in the activity “Derive other views”. As shown in Fig. 6.3, this activity consists at least of the derivation of the functional view, the information view, the operational view and the deployment view. Where needed, other views can be addressed. Examples of such views are the concurrency view, the enterprise view and the engineering view (Wikipedia 2013b; view model). As indicated in Fig. 6.3, this activity is contingent on the requirements process and it is also guided by the Physical Entity View and the IoT Context View. For instance, the IoT Context View might indicate that, due to the different ownership of parts of the system, a communication firewall is needed (→ functional view). In another example, the Physical Entity View might indicate that, due to the fragility of the Physical Entity, all devices attached need to be installed all at once (→ deployment view).

In order to accommodate different architecting methodologies, we have detailed the dependence of each of the actions in Fig. 6.3 on each other in the crib sheet in Table 6.1. This Table provides an overview of IoT architecting activities and actions (left columns) and what relevant input one derives from other IoT architecting activities and actions (horizontal).

Figure 6.3 gives a detailed view of the actions taking place within each activity (Create context view; Requirement process; Derive other views).

- In Red: actions that are particular to the IoT-A architecting framework and that directly contribute to the architecture documentation;
- In Orange: actions that are not unique to the IoT-A architecting process, but that enjoy an emphasis in the IoT-A framework;
- In Green: other activities and documents that directly contribute to the architecture documentation;
- In Blue: actions that are not unique to the IoT-A architecting process, but that enjoy an emphasis in the IoT-A framework
- <<flow>>: information flow into a document.

<i>View mapping of requirements</i>	requirements are covered.	View	requirements.	Mapping of design choices onto views.
<i>Derive other views</i>	Physical Entities and the basic relationships of said Entities.	First input on functional requirements, strategies for mitigating said risks.	Design choices pertaining to the functional view.	Requirements that are (partially) mapped onto functions.
<i>Derive information view</i>	Physical properties of interest.	Impact of risks and identified mitigation strategies on information and data model.	Design choices pertinent to information and data.	Identified FCs and the information to be exchanged between them.
<i>Derive operational and deployment view</i>	Physical Entities and the basic relationships and distributions of said Entities.	Implications of identified security risks on operation and deployment.	Design choices pertinent to deployment and operation.	Indications on operation from interactions view. Information on life cycle, distribution, and hierarchy of information.

Table 6.2 Overview of IoT architecting activities and actions

Architecting activity	Architecting action	Pertinent ARM module	Type of activity	Guidance/information provided in ARM module
Create IoT context view	Domain-model analysis	IoT domain-model (Sect. 7.3)	Create an IoT domain model of the envisaged IoT system. Use information provided in the context view for an identification of system boundaries and system scope	The IoT domain model including a thorough discussion of all its entities
Requirement process	Threat analysis	Reference manual pertaining to the IoT Domain Model (Sect. 9.1)		In-depth information on the entities in the IoT Domain Model together with examples of how to model these entities and entire systems
	Requirements engineering	Threat analysis (Sect. 4.1.7)	Perform a threat analysis for the envisaged system	List of system aspects to be taken into account during the threat analysis and what importance they carry
	Choose tactics	Design choices (Sect. 4.1.8)	Link identified security risks to design choices for mitigation of said risks	Lists of design choices for mitigating security threats to IoT systems
	Make design choices	How to use unified requirements (UNIs) (Sect. 6.7)	Generation of system requirements	Insight of how the high-level UNIs can support the generation of system requirements
		Perspectives (Sect. 8.3)	Mapping of requirements onto qualities (a.k.a. perspectives) and identification of suitable tactics for how to address said qualities. These tactics can be understood as architectural “lines of attack”	Minimum set of perspectives and associated tactics
		Design choices (Sect. 6.9)	By means of the tactics identified in the previous step, identify design choices for addressing the underlying qualitative requirements	Comprehensive taxonomy of tactics and associated design choices

View mapping of requirements	Non-functional views (Sect. 8.2.3 “Information View” and Sect. 8.2.4 “Deployment View”)	Perform first mapping of non-functional view requirements (and, where expedient, of qualitative requirements) onto the information view and the deployment and operational views	Overview of IoT-specific aspects of these views and how to model them
	Functional view (Sect. 7.2 “Functional Model” and Sect. 8.2.2 “Functional-View”)	Perform first mapping of functional requirements (and, where expedient, of qualitative requirements) to FCs. Identify interfaces needed	Description of FGs and FCs
Derive other views	Carrez et al. (2013)	Define basic functions, interfaces, and interactions for functional view	The Appendix Section provides exhaustive text on the FCs, related technical-use-case diagrams, descriptions of basic FC functions and high-level interfaces, as well as elementary interactions of FC and their basic functions (sequence charts)
	Chapter 10 “Interactions”	Define system-wide interactions of FCs	The Interactions Section provides examples of how such interactions can look like for a selected range of application scenarios
	Chapter 10 “Interactions”	Derive interaction patterns for prevailing usage scenarios	As briefly discussed in Sect. 8.2.2, the interaction viewpoint is an integral part of the functional view, but the IoT ARM is situated at a far too high level of abstraction in order to allow the definition of the functional interaction sequences at the reference-architecture level. In order to mitigate this systemic shortcoming of the IoT ARM, we decided to discuss some few usage scenarios of IoT systems that can shed at least some light on how

(continued)

Table 6.2 (continued)

Architecting activity	Architecting action	Pertinent ARM module	Type of activity	Guidance/information provided in ARM module
				such interaction patterns can look like, and they also provide inspiration for how such patterns might look like for the usages for which the concrete architecture is built
	Derive information view	Information view (Sect. 8.2.3 “IoT Information View”)	Define information and data model for all FGs and also for the exchange with applications and devices (see Sect. 8.2.3)	The IoT Information View provides several pieces of general information. First, it elaborates on general information descriptions (description of Virtual Entities service descriptions; associations between services and Virtual entities). Second, it elucidates the information handling of FCs both from an interaction- and from a interaction-scenario point of view. Third, it discusses the information life cycle
	Derive operational and deployment view	Deployment & operational view (Sect. 8.2.4)	Define how to deploy and how to operate the system	This section discusses in depth how deployment of IoT systems. Since the deployment patterns vary a lot, this Section of course focuses on common topics and patterns. First, it identifies other views and device descriptions as viewpoints for the deployment. This aids in translating information in these views into actionable deployment knowledge. This Section also discusses the question of connectivity solutions and where to host the

Derive deployment view	Carrez et al. (2013)	Define APIs for the implemented FCs	<p>services. It also touches upon the important questions of information storage and service resolution</p> <p>Carrez et al. (2013) provides definitions of high-level interfaces for all the FCs in the functional decomposition in Sect. 8.2.2 “IoT Functional View”.</p> <p>While these high-level interfaces are not equivalent to APIs, the information provided in Carrez et al. (2013) – (together with the sequence charts in the same Appendix) can be seen as a solid starting block for deriving APIs. More information can be found in the WP4 White Paper on Resolution Infrastructure Interface Binding (Bauer 2013), which describes a REST (Representational State Transfer) interface binding that was exemplarily defined based on the abstract interface specifications of the IoT Service Resolution and Virtual Entity Resolution FC found in Carrez et al. (2013)</p>
------------------------	----------------------	-------------------------------------	---

This table gives an overview of Activities and actions when architecting an IoT system and shows what modules of the IoT-ARM document one can consult in order to support said activities and actions. The Table also provides outlines about the input/guidance that can be derived from said modules. For an overview also see Fig. 6.4

6.5 IoT ARM Contributions to the Generation of Architectures

After the previous detailed overview of the architecting actions related to the generation of an IoT architecture, we are now finally ready for a discussion of how the IoT ARM contributes to the generation of specific architectures. As already outlined in Sect. 6.2, we do not prescribe a particular methodology for the generation of the architecture. The choice of a particular methodology is contingent upon factors such as the organisational structure of the architecting team, its “architecture history”, international standards or agreements that need to be adhered to, etc. Rather than prescribing a particular methodology and thus limiting its application range, the IoT ARM provides support and guidance for almost all of the actions and activities that are part of any architecting process (Fig. 6.4).

This figure gives a detailed view of the actions taking place within each activity and what parts of this document contribute to these activities and actions. The rectangular *dark-red boxes* represent sections in this document while <<flow>> represents information flow.

Table 6.2 discusses in more detail what each part of the IoT ARM contributes exactly to each of these actions and activities. This table also is intended to serve as a crib sheet for the architecting process.

6.6 Minimum Set of Functionality Groups

One question that we have often received concerns the least common denominator in terms of Functionality Groups of architectures that are derived from the IoT ARM. In other words: what Functionality Groups are part of any conceivable IoT ARM architecture?

The core aspects of IoT are things and communication. The things, i.e. Physical Entities (see Sect. 7.3) are accessed through devices, and data etc. pertaining to the Physical Entities is relayed by means of communication. Physical Entities are represented by Virtual Entities. Usually, the data is accessed via an application. Since we stipulate a service-oriented architecture framework in which the resources exposing data etc. about the Virtual Entities (and hence the Physical Entities) are exposed by IoT services, the minimum set of Functionality Groups is:

- Application Functionality Group
- IoT Service Functionality Group
- Communication Functionality Group
- Device Functionality Group

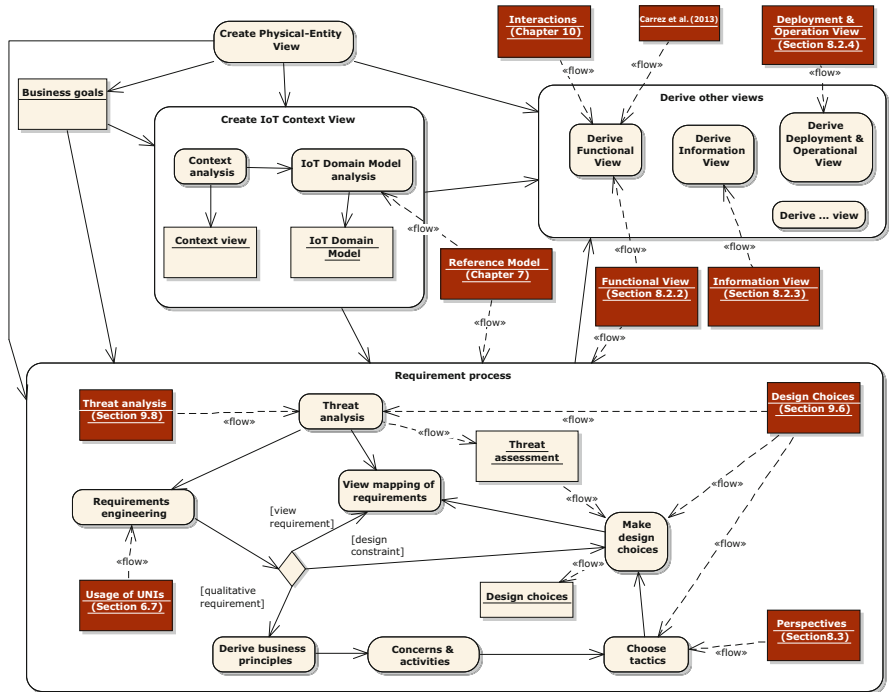


Fig. 6.4 IoT architecture generation (expansion of Fig. 6.3) summarises the parts of the IoT ARM that are relevant to the IoT architecting process and to what particular action

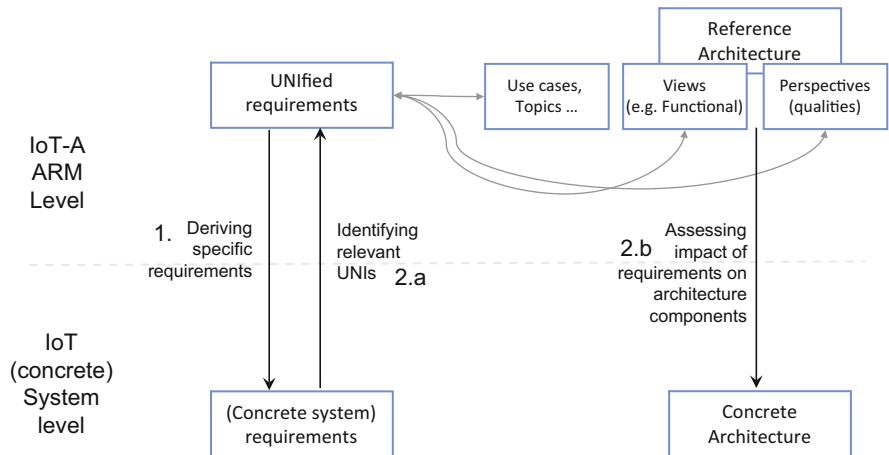


Fig. 6.5 Using IoT-A Unified Requirements and IoT ARM for concrete system architecture work

Note that this does not imply that other Functionality Groups (for instance, the Management Functionality Group) are optional. Rather, it means that for certain requirement sets these Functionality Groups are not needed.

6.7 Usage of Unified Requirements

6.7.1 Introduction

This section proposes guidelines to system architects on how to use the (already existing) Unified Requirement list (UNIs) during the Requirements process activity of their IoT architecture-generation process (Fig. 6.2). Such usage is by no means mandatory, as Requirement Engineering can be performed following the process described in Sect. 6.4 – however the UNIs list can serve as a helper tool to both the elicitation of requirements and to the system specification.

It is well known to system designers that requirement engineering is a crucial activity in system and software engineering. In the abundant documentation on the topic (e.g. Hull et al. 2011; Pohl 2010), one can distinguish three main steps where requirements play a role in designing complex systems: requirements elicitation (generally based on stakeholders input); deriving the system's specification from these requirements; and validating the implemented architecture.

As part of the work on the IoT Architectural Reference Model, UNIs were inferred and then published at <http://www.IoT-a.eu/public/requirements>. For more details on how these Unified Requirements were derived can be found elsewhere in the literature (Magerkurth et al. 2013). As these requirements do not apply to a concrete system, but rather to a Reference Architecture and a Reference Model applicable to all potential IoT systems, the reader needs to keep in mind a number of specifics before considering these Unified Requirements as input for the process of architecture translation:

- The Unified Requirement list should be seen as a basis and a living document. Although it tries to cover the whole spectrum of requirements families that could be applied to the IoT domain, it cannot be considered to be exhaustive, as, for instance, future regulation and legislation could impose requirements unforeseen at the time of publication. Additionally, Unified Requirements are often formulated on a quite high abstraction level (something largely avoided in concrete system's requirement engineering), resulting in requirements that are, for instance, mapped onto one or several views and possibly perspectives (again, something that concrete system designers tend to avoid);
- Formulation of requirements expressed by external or internal stakeholders (description field in the used Volere template) may sometimes apply directly to the IoT ARM (e.g. UNI.094 “The Reference Architecture shall support any IoT business scenario”), but in most cases they apply to a concrete system that can be implemented using the IoT ARM. In that latter case, they express characteristics on the system that the IoT ARM should enable to specify,

Table 6.3 Translation table for UNI requirement types from and to IoT ARM requirement types

UNI requirement type	IoT ARM requirement type	Indicated by
Design constraint	Design constraint	–
Functional requirement	View requirement	–
Non-functional requirement	View requirement	Mapping of UNI onto a view
	Qualitative requirement	Mapping of UNI onto one or more Perspectives

meaning they require to be interpreted by the reader/system designer to see how they apply to their own case – hence the wording “the system shall . . .” generally used. Let us take for instance UNI.021 “The user shall be able to control the radio activity of the system”: depending on the actual usage of radio communication, on the role of the user and on the importance of controlling the radio activity of the system in the concrete architecture, this requirement may be dropped, or specialised. In any case reinterpreting Unified Requirements is necessary (more on this in the following);

- Mapping to perspectives/views/functional groups and components is done on a lowest-common-denominator basis – e.g. it indicates which aspects are definitely impacted by a given Unified Requirement, but the reader should keep in mind that in certain (concrete system) specific cases, additional components may need to be considered. For instance, the Device Functionality Group is out of scope of the IoT ARM (see Figure “Functional-decomposition viewpoint” of the IoT Reference Architecture in Sect. 8.2.2.) and is therefore not listed in mapping of functional Unified Requirements, while it clearly needs to be considered when devising a concrete IoT system. Another instance is the lack of differentiation of the data plane vs. management plane in the IoT ARM, as this is a clear design choice (see Sect. 6.9).
- As pointed out in the ARM document, Sect. 6.4 and for the reasons explained there, the categorisation of the UNIs does not fully match that of the IoT ARM process and one needs to map the UNI categories onto that of the process in order to utilise the UNIs for the generation of architectures. Table 6.3 below provides this mapping information.

In a nutshell, the reader should keep in mind that the IoT ARM in general, and the Unified Requirement list in particular, should rather be seen as an inspirational than as a normative document.

6.7.2 Using Unified Requirements

IoT-A Unified Requirements (UNIs) can be used by system designers at two stages of their work: requirement elicitation and system specification.

6.7.2.1 Requirement Elicitation

UNIs can be used in a number of ways by system designers to identify “requirements topics” for their concrete system.

First, UNIs can be seen as seeds for deriving or instantiating concrete (precise) requirements from the broader, more abstract wording of Unified Requirements (Fig. 6.5- 1). For instance, UNI.018 reads “The system shall support data processing (filtering, aggregation/fusion, etc.) on different IoT-system levels (for instance device level)” (see Appendix). Based on this broad formulation, the system designer may derive his own requirements, identifying what kind of processing, on what kind of data, needs to happen where in his system.

Second, the mapping of the UNIs to Use Cases, facets of the IoT ARM (Models, Functional Groups, and Functional Components) or more informal categories can be used to filter and identify which topics and related UNIs should be considered by the system designer as potential candidates for instantiation on their own system. For example, using the web-based list, one can perform a global search on the word ‘communication’ (search all columns box), or filter all requirements categorised with the tag communication (Category column filter), or those which are sorted under the Communication Functionality Group (Functionality Group column filter) to see which UNIs in general apply to a given system.

6.7.2.2 System Specification

UNIs, and in particular their mapping to the IoT ARM, can also be useful to system designers during the specification phase. By identifying a UNI generalizing an already identified (concrete) system requirements (Fig. 6.5- 2.a), the various mapping on the IoT ARM enable the system designer to identify which IoT ARM components or more generally aspects are impacted by this requirement, and from there which concrete systems components or aspects need to be investigated (Fig. 6.5- 2.b). Figure 6.6 below presents this process using UML Activity diagram representation. Note that the “No corresponding UNI” case induces “regular” requirement engineering (i.e. without IoT ARM support).

For UNIs mapped on the Functional View, this enables the system designer to identify candidate functions in the concrete architecture that will be impacted by the overarching concern formulated in the UNI. For instance, UNI.623 reads “The system shall support location privacy”. This requirement is mapped on the Security and Privacy Perspective, which means that the system designer should consider this Perspective when deriving her own system requirements (more on this below). This UNI is also mapped onto four Functional Components in three different Functionality Groups of the Reference Architecture (namely IoT Service, IoT Service Resolution for the IoT Service FG; Authorisation for the Security FG; and VE Resolution for the Virtual Entity FG). After identifying how these FCs are instantiated (or not) in a concrete system, the system designer can use such a mapping to derive where the considered requirement(s) impact the concrete architecture.

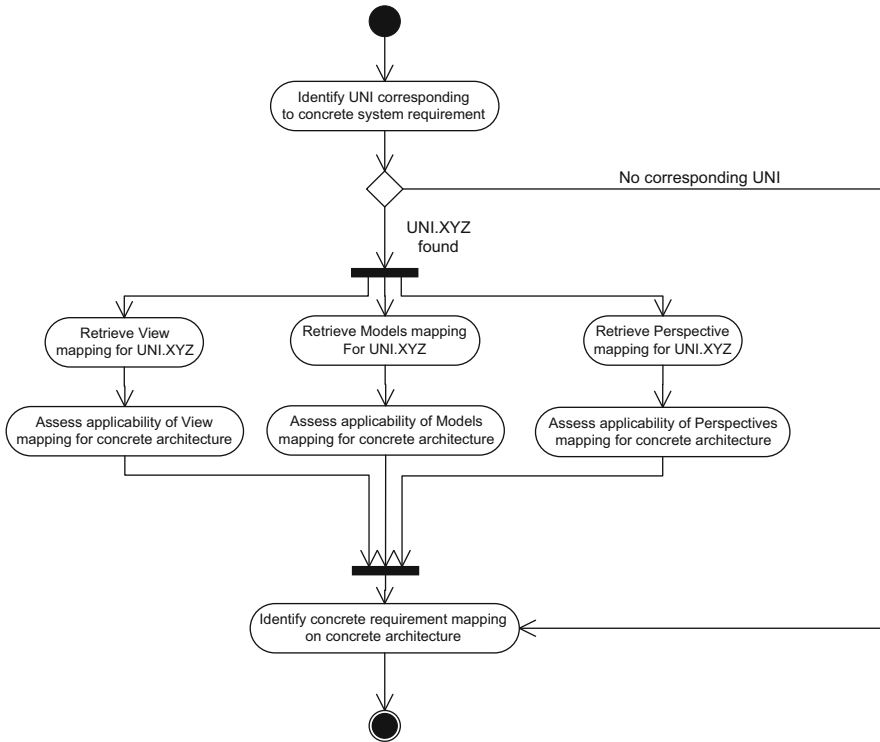


Fig. 6.6 How to use UNI to IoT ARM mapping to identify impacts of a given requirement on a concrete system architecture – activity diagram

Similarly, for UNIs assigned to quality aspects of the architecture (captured through ARM Perspectives), the mapping of UNIs onto design choices mapping (see Sect. 6.9 allows exploring perspectives and associated design choices that are impacted by a given UNI, and which therefore should be considered by the system designer. For instance, UNI.058 which reads “The system shall provide high availability” is mapped onto the Availability and Resilience ARM Perspective, and can be instantiated using two Design Choices (namely Cluster by location and Cluster by type of Resources). A corresponding concrete system requirement would typically provide more details (such as availability rate, etc.). After identifying which Perspectives apply (and how) to their concrete system, the system designer can use such a UNI-to-Perspective mapping to derive which quality aspects of the concrete architecture are impacted by the considered requirement(s).

6.8 Threat Analysis

As part of the setup of an IoT architecture, risk planning and resulting architectural decisions are of highest importance. The risk analysis carried out in this section aims therefore at assessing risks pertaining to the IoT, and at classifying them

according to the underlying mechanisms they apply to, the elements they affect, and the overall criticality they present.

Risk analysis traditionally begins with a definition of the elements that have to be protected. Then, an analysis of possible threats is conducted. How identified threats may actually affect elements to be protected, leads to the definition of risks. These risks have to be categorised, taking into account parameters such as criticality or probability of occurrence.

Various risk-analysis methods have been promoted in the literature, such as the French EBIOS (Ebios 2010) and OCTAVE (OCTAVE). The methodology for risk analysis that has been chosen in IoT-A, and that is used in this section, is based on Microsoft STRIDE/DREAD (Microsoft 2003). This choice has two reasons: first, this methodology is designed for assessing risks in the field of communications and information systems; second, it is mostly based on the analysis of architecture models and communications flows (instead of, for example, partly relying on experts interviews such as in EBIOS), which makes it a good fit for the ARM. The reasons for this are twofold. First, IoT, by its very name, encompasses information systems and communication. Second, no IoT-A-implementations are available at the time of writing. Therefore, the analysis has to centre on the Reference Architecture itself.

This section is organised as follows: first, a list of elements to be protected is provided. Then, the threats that may affect these elements (risk sources) are reviewed. The review follows the STRIDE classification. More details on STRIDE are provided below. The identified risks are then summarised and each risk is assessed in accordance with the DREAD methodology/metric.

This risk analysis is intended to be used as input for the derivation of architectures from the IoT ARM and for also for guiding the evolution of such architectures. By so doing one makes them more resilient against the most critical risks.

6.8.1 Elements to Protect

What elements need to be protected depends on the considered scenario. However, the IoT ARM was derived from the synthesis of a wide range of use-case areas, and identifying elements to be protected becomes rapidly very broad and multi-faceted. Instead, we decided to focus on the least common denominator of all use-case scenarios on which the IoT ARM is built. In other words, this analysis only looks at general elements to be protected, and this study is thus a good but non-exhaustive starting point for the study of a particular scenario to which the IoT ARM is going to be applied. The scenarios encompassed by the IoT ARM include:

- Transportation and logistics;
- Smart home;
- Smart city;

- Smart factory;
- Retail;
- eHealth;
- Energy (Smart Grid).

The following elements to be protected were identified:

- **Physical person:** This represents the human user. Threats affecting the human user are usually qualified as relating to ‘safety’ instead of ‘security’. Such threats may arise if a critical service is diverted or made unavailable by an attacker. An example for this is a malicious service that returns erroneous information, or even information specifically shaped to create hazardous situations. The eHealth scenario is the most critical concerning such attacks. Notice that the level of this criticality of course depends on the degree of automation. It is likely that most critical decisions will still require the involvement of a human operator;
- **Subject’s privacy:** This element represents all information elements that a subject (either a user or a device) does not explicitly agree to make publicly available, or whose availability shall be restrained to a controlled set of other subjects;
- **Communications channel:** The communication channel itself has to be protected. Common threats are attacks against the integrity of the data that are exchanged over the channel. Examples for such attacks are tampering and replay attacks. The communication channel shall also be protected against attacks aiming at the routing functionality of the underlying network (black hole, worm hole, depletion, etc.) (Mathur and Subbalakshmi 2007);
- **Leaf devices:** IoT-A leaf devices represent the wide variety of IoT elements that are interconnected by the common IoT-A infrastructure. Tags, readers, sensors, and actuators are examples for leaf devices. Various protection schemes relevant to their object class capabilities are to be implemented. These schemes need to ensure the integrity of the software, hardware, and the location of these devices;
- **Intermediary devices:** Intermediary devices provide services to IoT-A leaf devices and they also enable communication. A gateway designed to interconnect constrained and unconstrained domains is an example of such an intermediary device. Disabling or tampering critical intermediary devices can lead to denial-of-service attacks against the service infrastructure. Such attacks are within the scope of our analysis. However, attacks against specific intermediary devices that offer non-critical facilitating functions are outside the scope of our analysis and have thus to be considered case by case;
- **Backend services:** Backend services represent server-side applicative elements (for instance data-collection server communicating with sensor nodes). Compromising this software or the devices they are deployed on generally represents a critical threat against specific application systems and has to be prevented;
- **Infrastructure services:** Discovery, lookup and resolution services are very critical services as they provide worldwide fundamental functionalities to IoT systems. In the same way, security services (authorization, authentication,

identity management, key management, and trust and reputation) are essential for a secure interaction between subjects (as defined above);

- **Global systems/facilities:** This last category of elements to protect considers entire services in a global manner. For example, there might be a risk that an attack against the smart home scenario results in the complete disruption of the service, e.g. through the disruption of underlying communications between devices. The consequences of this resulting disruption can therefore be considered through this category.

6.8.2 Risk Sources

The risk sources are categorised following the STRIDE (Microsoft 2003) classification, which is a widely used way of classifying threats that relate to information systems. STRIDE stands for **S**poofing identity, **T**ampering with data, **R**epudiation, **I**nformation disclosure, **D**enial of service, and **E**levation of privilege. These categories are quickly summarised below – note, however, that real-world occurrences usually consist of a combination of these threats.

- **Identity spoofing** means that a peer illegitimately uses the identity of another peer. Spoofing attacks can happen with respect to all kind of identifiers, irrespective of whether they are used to designate physical persons, devices, or communication flows;
- **Data tampering** means that an attacker is able to alter the content of data exchanged between two or more peers. Data tampering may involve subtle attack schemes, wherein the attacker is able to trigger specific behaviours of recipients by finely modifying original data;
- **Repudiation** relates to attacks in which an attacker performs illegitimate actions and may afterwards deny having performed them, such that other nodes are unable to prove that the attacker actually behaved maliciously;
- **Information disclosure** means that information is disclosed to unauthorised peers. It is related to the existence of an authorisation model that defines for each information element a set of peers that are authorised to access it, possibly under some specific conditions;
- **Denial-of-service** attacks are carried out for disabling a service offered to legitimate users (as opposed, for example, to more subtle schemes wherein the attacked service can be altered, e.g. making a search service return false results, without the legitimate users being able to notice it);
- **Elevation of privilege** may occur in systems that feature different classes of users, each class being mapped to a specific set of rights. Illegitimate elevation of privilege occurs when an attacker manages to acquire rights that would normally only be granted to more privileged class(es). In the most critical case, an attacker may obtain administration rights for the entire system, or part of it, which means that the attacker may perform arbitrary actions on the elements the attacker has access to, thereby being able to destroy the system or entirely change its behaviour.

The risk sources considered here are restricted according to the following rules:

- Non-human risk sources either global (flood, lightning, fire, electrical, heat) or local (individual device failure) are not considered. Only human risk sources are. Note that a human forging a faked device identity in order to impersonate another device fits within the category of “human risk”;
- Among human risk sources, only theft/loss and hacker-initiated attacks are considered. Technical staff errors or accidents are not considered. In other words we are only addressing malicious attacks and not involuntary attacks.

The STRIDE classification is used below in Table 6.4, immediately afterwards, on STRIDE classification] to identify risks, as intersections between a STRIDE item (column) and an element to protect (row).

6.8.3 Risk Assessment

Identified risks were assessed using the DREAD methodology based on (simplified) metrics. DREAD, defines scoring methodology and metrics that help to evaluate the criticality of an identified threat. DREAD stands for **D**amage potential, **R**eproducibility, **E**xploitability, **A**ffected users, and **D**iscoverability. It defines the criteria according to which a threat is evaluated. Each criterion is quantified at levels between 0 and 10. Eventually, the threat can be globally rated (sum of D, R, E, A, D ratings), or the threat can be described along with its individual ratings. The latter approach allows, obviously, for a more precise analysis. A simpler scheme for DREAD, used in what follows, consists of only three levels, viz. L (low), M (medium) and H (high) for each DREAD rating.

Note that a ‘High’ rating for Exploitability means that it is easy for an attacker to carry out an attack leading to the identified threat, whereas a ‘High’ rating in Discoverability means that it is difficult to discover the threat. This is to ensure a coherent approach, in which ‘Low’ ratings decrease the overall criticality of a risk, whereas ‘High’ ratings increase it.

The DREAD methodology and metric is used in Table 6.5, immediately afterwards, on DREAD assessment] for evaluating the risks identified in Table 6.4, the previous one, on STRIDE classification]. In addition to the DREAD rating, the Table 6.5 on DREAD assessment]also provides initial information on specific threats that may lead to the occurrence of the identified risk. In addition to this information, initial steps toward threat mitigation are provided. Furthermore, it links mitigation scenarios to the design choices (noted DC X.n) elaborated on in Sect. 6.9.

6.8.4 Discussion

Assessing the risks that relate to the Internet of Things and putting them in perspective with the Design Choices (see Design Choices) leads to interesting

Table 6.4 STRIDE classification (horizontal) of the identified risks broken down by the elements to be protected (vertical)

	Spooing identity	Tampering with data	Repudiation	Information disclosure	Denial of service	Elevation of privilege
Physical person		Attack alters data so that wrong data is supplied to a critical monitoring system	Human users might use unattended electronic devices without leaving a digital trace		A service critical for user's safety is disabled	
Subject's privacy	User's identity is spoofed			Attacker gleans knowledge of user private parameters		
	User is involved in transactions with a malicious peer			Attacker gleans knowledge of user's location		
Communication channel		Alteration of the invocation of a service	Jamming wireless communication channels leads to local denial-of-service attacks that can be repudiated (no digital traces)	Attacker gains knowledge of sensitive exchange data	Attacker disrupts communications	Wrong authorisation information propagating from one server to another
Leaf devices	Loss or theft of physical device used for authentication	Alteration of the return value upon service invocation		Disclosure of device configuration information	Attacker physically disables local leaf device	
		Attacker gains control of an actuator		Device identification may divulge sensitive information, or may be linked so that it	Attacker physically disables remote leaf device	
	Attacker alters leaf-device content so that a user will eventually be					

<p>Intermediary devices</p>	<p>Attacker changes the association between a Virtual Entity and the corresponding Physical Entity</p>	<p>redirected to a malicious content</p> <p>Attacker alters sensor device so that monitoring of a Physical Entity fails</p>	<p>exhibit information about usage patterns,</p> <p>Loss or theft of device containing private information</p> <p>Attacker prevents proper communication to an actuator</p>
<p>Backend Services</p>	<p>Usurpation of administrator role</p> <p>Backend account hacked</p>	<p>Compromised intermediary devices alter traversing data</p> <p>Intermediary devices behave maliciously and clients are not able to report the fact</p> <p>Information re-routing by intermediary device so that it ends up at an unintended destination</p> <p>Massive disclosure of collected data</p> <p>Backend service is made unavailable</p>	<p>Assisting intermediary devices are no longer usable</p>
<p>Infrastructure services</p>	<p>Attacker impersonates services and compromises IoT functionalities and/or other dependent infrastructure services</p>	<p>Attacker poisons infrastructure databases and/or alters outgoing information</p>	<p>Attacker denies legitimate users access to infrastructure services</p>
<p>Global systems/facilities</p>	<p>Disclosure of access policies</p> <p>Disclosure of Identities and cryptographic material</p> <p>Massive disclosure of user information</p>	<p>Disclosure of access policies</p> <p>Disclosure of Identities and cryptographic material</p> <p>Massive disclosure of user information</p>	<p>Disruption of a global service</p>

Table 6.5 DREAD assessment of the identified risks (see Table 6.4 STRIDE classification)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
Physical person	Attack alters data so that wrong data is supplied to a critical monitoring system	H/L/M/L/L →enforce strong security		Data-integrity protection provided as part of protocol security. DC S.16: cryptographic protocols DC S.19: integrity protection obtained from authentication enforcement at link layer
	Human users might use unattended electronic devices leaving no digital trace	L/L/H/L/L →enforce weak security		Addressable through proper (local/remote) user authentication scheme which is a feature of the Authentication Functional Component (see Sect. 8.2.2.7). DC S.1,3: ensure proper logging of authentication operations, e.g. through the use of a AAA (authentication, authorisation, and accounting) or a AAA-like system
	A service critical for user's safety is disabled	H/M/M/L/L →enforce strong security		Critical services have to be protected through redundancy of their key elements. Malicious actions are prevented through dedicated access-control policies (security management). Communication medium between user and critical service has to be made robust against DoS attacks at all OSI layers DC S.5: restrained service access DC A.16–17: autonomous security

User's privacy	User's identity is spoofed	L/H/H/L/M →enforce strong security	Credential theft	Robust user-authentication procedure preventing man-in-the-middle attacks, with proper credentials management policy provided by Authentication Functional Component (see Sect. 8.2.2.7)
			Credential brute-forcing	DC S.1: authentication over encrypted channel DC S.10 avoid common crypto credentials; avoid reliance on symmetric crypto
			Registration procedure that is vulnerable to man-in-the-middle attack	
	User is involved in transactions with a malicious peer	L/H/H/M/L →enforce strong security	Redirection to malicious content. The redirection may be caused by data tempering on communication channel or leaf node compromising (e.g. content of a tag is altered)	Trustworthy discovery/resolution/lookup system. Trustworthiness of the entire system is enabled through its security Functional Components (especially Authentication and Trust and Reputation (see Sect. 8.2.2.7), as well as its global resilience to intrusions (security by design)
	Attacker gains knowledge of user configuration	M/M/M/L/H →enforce strong security	User's private information leakage through user's characterisation as requiring certain data (and thus performing accordingly discovery, lookup, resolution of the corresponding services)	Resolution security DC S.1: authentication over encrypted channel Enforcement of a robust pseudonymity scheme ensuring both anonymity and unlinkability of two successive data units; provided by the Identity Management Functional Component (see Sect. 8.2.2.7)

(continued)

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
			<p>User's private information leakage through user's characterisation as providing certain data.</p> <p>Traceability (this path, hence this user).</p>	<p>DC S.10: encryption schemes, with a specific relevance of onion-routing-like encryption (best scheme with respect to anonymity support)</p> <p>DC P.1: temporary identity, more easily changed for unlinkability, hence privacy</p>
	Attacker gains knowledge of user's location	L/H/M/L/H →enforce weak security		<p>User's location can be hidden through reliance on pseudonyms provided by the Identity Management Functional Component (see Sect. 8.2.2.7)</p>
Communication channel	Alteration of the sent invocation of a service	L/L/M/L/L →enforce weak security		<p>DC P.1: temporary identity, more easily changed for unlinkability, hence privacy</p> <p>End-to-end integrity protection of service-access signalling (data integrity protection is provided as part of protocol security)</p>
	Alteration of the return value upon service invocation	L/L/M/L/L →enforce weak security		<p>DC S.1.3: service-based data integrity</p> <p>DC S.19: integrity protection obtained from authentication enforcement at link layer</p> <p>End-to-end integrity protection of service-access signalling (data integrity protection is provided as part of protocol security)</p>

<p>Jamming wireless communication channels can lead to local denial-of-service attacks that can be repudiated</p>	<p>M/H/L/M/M →enforce medium security</p>	<p>Service-based data integrity DC S.19: integrity protection obtained from authentication enforcement at a layer below the service Jamming denial-of-service attacks can be addressed through physical means: for instance, once the attack is detected localise and neutralise the jammer DC A.16–17: autonomous security <i>could</i> be enabled for detecting this attack</p>
<p>Attacker gains knowledge of sensitive, exchanged data</p>	<p>M/L/M/L/L →enforce medium security</p>	<p>End-to-end confidentiality protection of exchanged data, offered through protocol security DC S.10: encryption schemes</p>
<p>Attacker disrupts communications</p>	<p>M/H/L/H/L →enforce medium security</p>	<p>Various denial-of-service prevention schemes are available. Their applicability depends on the communication technology used (anti-jamming, enforced MAC, etc.). Schemes are offered through security-by-design of the communication stack DC A.16–17: autonomous security systems are generally able to deter denial-of-service attacks, however lightweight schemes are less powerful</p>

(continued)

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
	Wrong authorisation information propagating from one server to another	M/L/L/H/M →enforce medium security		Strong security for server-to-server communications that leverages individual's credentials (e.g. certificates) instead of group keys, and allows for revocation (security by design, adequate management policies)
Leaf device	Loss or theft of a physical device used for authentication	M/L/H/L/L →enforce weak security		Two-factor authentication, when applicable. This means that the gain of the physical device would not be enough for an attacker to pretend being a legitimate user and authenticate as such Cryptographic credentials should be themselves protected (PIN code, passphrase)
	Loss or theft of physical device containing private information	M/L/H/L/L →enforce medium security		DC S.1.3: authentication. Note that identification instead of authentication should <i>not</i> be applied Physical protection of stored credentials (e.g. security vault) – readability of a device only upon fulfilment of certain conditions (e.g. known reader)

<p>Attacker changes the association between a Virtual Entity and the corresponding Physical Entity</p>	<p>M/L/M/H/L →enforce medium security</p>	<p>Wrong tag on a device</p>	<p>Secured discovery/resolution/lookup system A specific Design Choice for tamper-proof IDs is not provided for two reasons. First, one could realise it on a hardware-level by using tamper-proof hardware modules. Notice that hardware is out of scope for IoT-A (device level is not part of the RA). The second reason is that tamper-proof IDs can also be realised by a secure resolution system by means of Authentication and Authorisation which is already part of the RA and thus no Design Choice is needed</p>
<p>Attacker gains control of an actuator</p>	<p>Compromising resolution system M/M/M/L/M →enforce medium security</p>		<p>Proper authorisation scheme as offered by the Authorisation Functional Component (see Sect. 8.2.2.7) End-to-end integrity protection, provided as part of protocol security DC S.5: prevent compromise through access restriction DC A.16–17: reactive (autonomous) security in case of compromise Not specifically targeted. Addressable through a proper URI verification system on user device</p>
<p>Attacker alters leaf-device content so that a user will eventually be redirected to a malicious content</p>	<p>M/M/H/M/L →enforce medium security</p>		

(continued)

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
Attacker alters sensor device so that monitoring of a Physical Entity fails	L/M/L/L/H →enforce weak security	L/M/L/L/H →enforce weak security		Not specifically targeted. Sensitive physical values may be monitored by a large number of sensors, or sensor integrity can be remotely verified
Disclosure of device configuration information	L/L/L/L/H →enforce weak security	L/L/L/L/H →enforce weak security		Not specifically targeted. Unlinkability between different actions of the same device, provided by the Identity Management Functional Component (see Sect. 8.2.2.7), will mitigate the criticality of this threat DC P.1: use of temporary identity to provide unlinkability
Device identification	L/M/M/L/H →enforce medium security	L/M/M/L/H →enforce medium security	Attacker bypasses in-place pseudonymity scheme and identifies a device as providing access to certain data	Adequate protection scheme requiring partial pre-knowledge of each other before a tag can be read by a reader (the tag will only answer to a “known” reader)
Attacker physically disables leaf device (local)	L/H/H/L/L →enforce weak security	L/H/H/L/L →enforce weak security	Tag destruction	Not specifically targeted. Typically addressable through physical investigation (identify the attacker through traces left by the physical attack; e.g. triangulation of a destructive electromagnetic pulse)

<p>Attacker physically disables leaf device (remote)</p>	<p>M/H/L/H/L →enforce weak security</p>	<p>Tag destruction by remote electromagnetic means</p>	<p>Not specifically targeted. Typically addressable through physical investigation</p>
<p>Attacker prevents proper communication to an actuator</p>	<p>M/H/L/M/L →enforce medium security</p>		<p>Denial-of-service detection/reaction scheme (security by design)</p>
<p>Compromised intermediary devices alter data passing through</p>	<p>M/H/M/M/L →enforce medium security</p>		<p>End-to-end security scheme provided by the Key Exchange and Management Functional Component (see Sect. 8.2.2.7), and enforced by the relevant protocol security function</p>
<p>Intermediary devices</p>			<p>Remote monitoring of intermediary devices can be another means of dealing with this threat (identification of compromised devices)</p>
<p>Intermediary devices behave maliciously and clients are not able to report the fact</p>	<p>M/M/L/M/H →enforce weak security</p>		<p>DC S.10: end-to-end encryption DC A.16–17: autonomous security Remote monitoring of intermediary devices</p>
			<p>Depending on the malicious action performed by intermediary devices, client nodes may mitigate it by applying end-to-end security schemes (Key Exchange and Management Functional Component + protocol security)</p>
			<p>DC A.16–17: autonomous security</p>

(continued)

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
	Information re-routing by intermediary device	M/H/M/M/M →enforce medium security		End-to-end security scheme put in place by the Key Exchange and Management Functional Component (see Sect. 8.2.2.7), and enforced by the relevant protocol security function
	Assisting intermediary devices are no longer usable	L/M/H/H/L →enforce medium security	Exhaustion attacks	DC S.10: end-to-end encryption Denial-of-service detection/reaction scheme DC A.16–17: autonomous security
			Various specific attacks against the involved assistance mechanisms (e.g. no packet forwarding toward a routing service, replacing a received key fragment with garbage against a collaborative keying service...)	
Backend services	Administrator-role usurpation	H/M/L/H/L →enforce medium security	Administrator credentials disclosed/hacked/brute-forced	Not specifically targeted. Addressable through security management and credentials management policies
	Backend account hacked	M/M/L/H/M →enforce medium security		Not specifically targeted. Addressable through security management and credentials management policies
	Massive disclosure of collected data	H/M/L/H/L →enforce medium security		Not specifically targeted. Addressable through security management (databases)

<p>Backend service becoming unavailable</p>	<p>L/M/M/H/L →enforce medium security</p>	<p>DoS detection/reaction scheme DC A.16–17: autonomous security</p>
<p>Attacker impersonates infrastructure services, compromising IoT functionalities and/or other dependent infrastructure services</p>	<p>H/M/L/H/M →enforce strong security</p>	<p>Prevention of impersonation techniques through proper use of authentication/authorisation procedures (enforced by the respective Authentication and Authorisation Functional Components (see Sect. 8.2.2.7)</p>
<p>Attacker poisons infrastructure databases (records corruption/addition) or alters outgoing information</p>	<p>H/H/L/H/M →enforce strong security</p>	<p>DC S.1.3: authentication Proper authorisation scheme provided by the Authorisation Functional Component (see Sect. 8.2.2.7) mitigates this attack. Enforcement of a trust model (Trust and Reputation Functional Component (see Sect. 8.2.2.7) protects against blind acceptance of erroneous data DC S.5: service access control.</p>
<p>Disclosure of private services (existence & description)</p>	<p>L/H/H/M/M →enforce medium security</p>	<p>Although this does not allow identifying corrupted data, it may help identifying and excluding the attacker Masking the belonging of multiple services to a single entity (unlinkability). This can be achieved by reliance on pseudonyms provided by the Identity Management Functional Component (see Sect. 8.2.2.7) DC P.1: mitigation through the use of temporary identifiers</p>

(continued)

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
Disclosure of access policies		L/H/H/M/M →enforce medium security		Security management of infrastructure prevents global disclosure of access policies from the decision point to an unauthorised external attacker. Probe discovery of access policies by authorised, though compromised internal attackers are more subtle, and have to be dealt with through adaptive security (e.g., recognise a malicious pattern in the regular probing of security decision points)
Disclosure of identities and cryptographic material		M/H/H/M/L →enforce strong security		DC A.16–17: probing detection/reaction performed by autonomous security
Attacker denies legitimate users access to Infrastructure Services		M/H/L/M/L →enforce medium security		Not specifically targeted – addressable through security management (databases)
				Exclusion of the attacker, once identified as such through the Trust and Reputation security Functional Component (see Sect. 8.2.2.7)

Massive disclosure of user's personal information

H/L/L/H/L
→enforce **strong** security

Secure storage of personal data with dedicated protection architecture (e.g. firewall diodes that let data flow in one direction only) and access control rules—this is part of security management

Disruption of a global service

H/M/L/H/L
→enforce **strong** security

Reliance on all security Functional Components (see Sect. 4.2.2.7) + proper security management
This threat can also be addressed by multihoming. See **DC P.3** (Replication of instances of Functional Components locally) and **DC P.4** (Replication of instances of functional components in the cloud)

synthetic conclusions. First, we recognise in the risks and their mitigation mechanisms the well-known distinction between **internal attacks** and **external attacks**. This distinction implies the existence of a discrimination function that makes the system able to distinguish among authorised players (hence, able to launch internal attacks) and unauthorised players (restrained to external attacks). Second, it is also noticeable that some risks are not mapped to design choices – rather, they can be mitigated through dedicated context-dependent or local (entity-scope) security-by-design decisions. These concepts are elaborated on in what follows.

The distinction between internal and external attackers pertains to their ability to undergo an authorisation procedure, at the end of which only authorised players acquire some rights. These rights in turn enable the attackers to launch internal attacks. Note that this authorisation procedure may be characterised by more than the rejected/authorized two levels of granularity and define a full set of access policies. In this case, all but entirely rejected players are in position to launch internal attacks.

The defence against **external attacks** is traditionally based on two means: topological defence systems that almost *spatially* keep the attackers out of reach of the protected resources (e.g. firewalls) and cryptographic mechanisms (e.g. authentication or encryption algorithms) that *logically* prevent attackers to tamper with or otherwise access the protected resources.

- In the framework of IoT, special emphasis is put on one-to-one transactions wherein a service is accessed by a remote player. These transactions require a **secure transaction set up**. The service-access control involves in its most secure embodiments an authentication phase that can be based on various authenticating credentials. It has to be noted, though, that these authenticating credentials have to be mapped to an identity in order to fulfil their role. When the peer identity is not known prior to establishing a transaction, it has to be securely retrieved (resolved) from the resolution infrastructure. Likewise, the services themselves may need to be securely orchestrated;
- Upon successful authentication, **access control** has to be enforced in order to bind all data units exchanged between two players to their respective authenticated identities. This takes usually the form of an authentication procedure being implemented as an *authenticated key-exchange* (AKE) protocol, and all subsequent messages exchanged between the same two players are then integrity protected by the AKE-obtained session key. Various protocols exist for doing so: at the network layer, the *Host Identity Protocol Base Exchange* (HIP BEX) and *Internet Key Exchange* (IKE) are AKE protocols and IPsec is the corresponding secure data transport protocol. At the transport layer, TLS handshake is an AKE protocol for subsequent (D)TLS exchanges. Various service-specific protocols can of course also be used. Eventually, all risks mitigated by integrity protections should rely on specific cryptographically protected access-control schemes;
- In parallel with secure transaction set up and access-control-based integrity protection, protection against internal attacks requires a **coherent arrangement**

of the associated cryptographic primitives which have to be based on an assessment of the attacker profile and capabilities. Many design choices proposes different embodiments that provide different security levels. For example the perfect forward secrecy property is theoretically a more secure one. However, this additional security property would prove worthwhile only for an attacker able to (and interested in) accessing data exchanged in the past (hence possibly obsolete) but that the attacker would nevertheless have stored under an encrypted form. Clearly, most of attacker models and data criticality do not fit within this attack scenario. If one decides to envision it, though, the same attacker capabilities should be assumed for all other risks.

Protection against **internal attacks** is illustrated in the Table 6.5 on DREAD assessment by the reliance on **autonomous security** design choices (DC A.16,17). Classically, only behavioural analysis can allow identifying misbehaviours of an otherwise authorised node. Autonomous security can be instantiated under a wide variety of forms that pertain to the implemented functions in a given IoT infrastructure. Whenever behavioural patterns can be defined, deviations from these patterns can be detected and flagged as suspicious. More generically (and more easily), **logs** should be enabled as a rudimentary form of reactive security. Logs can be generated at various places in the network but will generally be aggregated at server-side, where they will be collected for further uses such as service management (e.g. dimensioning), lawful requirements or billing preparation. However, logging user activity or detecting identifying patterns within it countervails privacy. Autonomous security and privacy are in general mutually contradictory. Pseudonymity can be seen as an intermediary state, although pseudonyms are only worthwhile as long as they can be resolved to real identities at some point in the network. Choosing which scheme to favour is a question of high-level design choice. Diametrically opposed to privacy, non-repudiation plays a specific role that has to be reviewed here. In general, this security service, which ensures that an entity will not be in position of denying having performed a given transaction, is provided at service layer where both signature-based cryptographic primitives and transaction concept become relevant. Although the associated risk (repudiation) is part of the STRIDE classification, service-level non-repudiation was not considered in the previous section, being judged to be pertaining to policies, themselves associated to particular applications. In fact, services for which non-repudiation has to be provided are part of highly specific applications (e.g. inter-bank communications of aggregated banking transactions, or administration of highly-critical assets), which does not qualify them as generic mitigation means.

Finally, it is worth explaining why some identified risks are “not specifically targeted” in IoT-A, with no relevant technology being developed and no design choice being proposed. These non-targeted risks are of two sorts. Some of them are dependent on highly contextual physical parameters. They depend on the particularities of the communication technology that is put in place and, as such, exhibit highly diverse characteristics in terms of involved stakes. Accordingly, the existing mitigations can only be implemented at the physical layer with variable

costs in terms of, for instance, efficiency. The other non-targeted security risks pertain to in-entirety security-by-design policies. For example, the protection of a given operating system or the choice to encrypt a user database fit into this category. As such, they cannot be qualified as being typical for the IoT environment.

6.9 Design Choices

6.9.1 Introduction

By following the architectural methodology according to (Rozanski and Woods 2011) it is recommended to apply the architectural perspectives to the views on an architecture in order to design systems that satisfy qualities like high performance, high scalability or interoperability. This step in the architectural methodology is similar to constructing the interrelationships between customer requirements and technical requirements in the ‘House of Quality’ matrix as applied in the Quality-Function Deployment (Erder and Pureur 2003) introduced in Sect. 6.4.

This section guides an architect by giving design choices for the architectural viewpoints defined in the Reference Architecture in Section 8.2 for each perspective listed in Sect. 8.3. Figure 6.7 illustrates that the perspectives ‘Evolution & Interoperability’, ‘Performance & Scalability’, ‘Trust, Security & Privacy’, and ‘Availability & Resilience’ are applied to the ‘Functional View’, the ‘Information View’ as well as the ‘Deployment & Operation View’ respectively.

While applying perspectives to views not every view is impacted by the perspectives in the same manner or grade. Rozanski and Woods distinguish between three grades of applicability (high, medium and low) for each perspective to each view. Table 6.6 illustrates the perspective to view applicability as presented in (Rozanski and Woods 2011).

In this section we focus mainly on the perspective and view pairs where the applicability of the perspective to the view is high. According to the Table 6.6 these pairs are the following:

None of the perspectives have a high impact when applied to the Operational View. This is an indicator for not considering the Operational View in the RA (Sect. 8.2) and therefore in this section respectively. The Concurrency View is not being considered in the RA Sect. 8.2 either, thus the applicability to this view, even with a high impact, is not followed up in this section.

Additionally, we do not present design choices for particular platforms (i.e. recommendations for specific hardware and software) as they would give the current status of available platforms at the time of editing this document only, but the recommendations could become obsolete soon after. Software architects are well advised to look for suitable platform solutions while designing their concrete architectures. Platforms that were researched during the project (Magerkurth 2011) are based on the OSGi framework (OSGi 2012). This framework specifies among others how software can be deployed in form of bundles and how the application lifecycle can be controlled remotely. The OSGi framework is a recommended

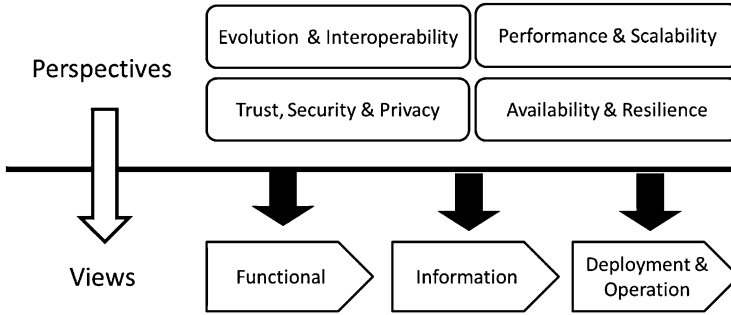


Fig. 6.7 Applying perspectives to views (Rozanski 2011; Fig. 6.4–1)

Table 6.6 Typical view and perspective applicability (Rozanski and Woods 2011)

Perspective \ View	Security	Performance & Scalability	Availability & Resilience	Evolution
Functional	Medium	Medium	Low	High
Information	Medium	Medium	Low	High
Deployment	High	High	High	Low
Operational	Medium	Low	Medium	Low
Concurrency	Low	High	Medium	Medium

Table 6.7 Focus on high perspective to view ability

Architectural perspective	Architectural view
Evolution and interoperability	Functional Information
Availability and resilience	Deployment
Performance and scalability	Deployment
Trust, security and privacy	Deployment

design choice for the Deployment and Operation View. Based on the experience obtained in the project we recommend OSGi framework as a design choice for the Deployment and Operation Views with hardware platforms that provide support for OSGi. However, OSGi framework is not advisable for very constraint computing platforms.

According to Rozanski/Wood “a tactic is much more general and less constraining than a classical design pattern because it does not mandate a particular software structure but provides general guidance on how to design a particular

aspect of your system” (Rozanski and Woods 2011). Following Rozanski and Wood’s definition this section picks up the tactics addressing the architectural perspectives listed in Sect. 8.3 and presents technology agnostic design patterns or other architectural solutions that are suitable to apply the tactics. Architects are then able to either implement the recommended design choices or to look for existing solutions that have implemented those choices.

6.9.2 Design Choices Addressing Evolution and Interoperability

The Evolution perspective addresses the fact that requirements change and software evolves sometimes rapidly. We identified a second, closely related, perspective namely Interoperability which plays a crucial role especially in IoT. The vision of the Internet of Things is still evolving. Many current technologies are not yet mature enough for operational use and there are many more technologies to come in the future. The Evolution and Interoperability Perspective is shown in Sect. 8.3.1. The tactics for evolution and interoperability are the key concepts of the IoT ARM and will be explained in Table 6.8.

Both, the Reference Model and the Reference Architecture are built to be extensible and to enable interoperability between Devices and Services. Therefore the activities listed in Sect. 8.3.1 reflect the IoT-A approach in detail:

- **Characterize the evolution needs:** IoT-A has collected stakeholder and also internal requirements reflecting the actual and future needs in IoT systems (see IoT-A 2013);
- **Assess the current ease of evolution:** Also through the stakeholder workshops and in addition the use cases from WP7 and the state of the art analysis from WP1 and all technical work packages, the current status was collected;
- **Consider the evolution trade-offs:** The evolution trade-offs are heavily domain- and application-specific and are not part of the IoT-A work. Those trade-offs must of course be discussed when creating an architecture for a concrete application;
- **Rework the architecture:** The main result of IoT-A are the Reference Model and the Reference Architecture which were designed with interoperability in focus (see Sect. 7.5 and Chap. 8).

Moreover, Rozanski and Woods (2011) also introduce tactics to deal with interoperability and evolution. Here also the IoT-A Reference Model and Reference Architecture adapt the following tactics:

- **Create extensible interfaces, Apply design techniques that facilitate change:** IoT-A defines common entities, e.g. the IoT Domain Model, see Sect. 7.3, and entry points, e.g. the IoT Communication Model, see Sect. 7.6, which can be used to create IoT-A compliant systems;

Table 6.8 Tactics addressing evolution and interoperability

Desired quality	The ability of the system to be flexible in the face of the inevitable change that all systems experience after deployment, balanced against the costs of providing such flexibility
Tactics	Create extensible interfaces Apply design techniques that facilitate change Apply metamodel-based architectural styles Build variation points into the software Use standard extension points

Table 6.9 Tactics identified as not relevant for evolution and interoperability in IoT Systems

Tactic	Reason
Contain change	Not possible for public IoT-systems, new devices will participate in the systems
Achieve reliable change	Same as above
Preserve development environments	Due to the multiplicity of developers and technology providers, a common development environment will not exist

- **Apply metamodel-based architectural styles:** The IoT-A Reference Model and Reference Architecture define interoperability on architectural level. Especially the Domain Model, see Sect. 7.3, and the IoT Information Model, see Sect. 7.4, as metamodels are open for further extensions;
- **Build variation points into the software, Use standard extension points:** By using standardised protocols and gateways, even legacy devices are able to be linked to IoT-A systems.

Design Choices for Interoperability and Evolution cannot be named on this (application and domain independent) level. The IoT Reference Model and Reference Architecture are built with interoperability and evolution as the main drivers. To allow a system to evolve and to react to new technology and new requirements the following general remarks should be kept in mind:

- The IoT-A Reference Architecture is built out of modular blocks to allow changes and additions. When deriving the IoT-A work to a concrete architecture, this modularity and also the loose coupling between those blocks should be kept. This concept is also used in the ‘Dispatcher’ component (Hyttinen P ed et al. 2013) for the standardized processing of incoming requests without exposing the internal methods and functions;
- Not all of the systems functionality can be defined in advance. Therefore, some additional spaces and extensions points, e.g. for upcoming functionality, should be reserved. This can for example be done in interface definitions or data models, like the reserved bits in the TCP header definition. This allows the designers and architects to update the system and to adapt it to new requirements.

The tactics not considered as relevant are listed in Table 6.9.

Table 6.10 Tactics addressing performance and scalability

Desired quality	The ability of the system to predictably execute within its mandated performance profile and to handle increased processing volumes in the future if required
Tactics	Optimize repeated processing Replication Prioritize processing Distribute processing over time Minimize the use of shared resources Reuse resources and results Partition and parallelize Scale up or scale out Degrade gracefully Use asynchronous processing Reduce complexity Make design compromises

Table 6.11 Tactics and corresponding design choices for performance and scalability

Tactic	Impact on views		
	Functional	Information	Deployment and operation
Replication	Replication of functional components (DC PS.1)	Replication of gathered Information (DC PS.2)	Replication of instances of Functional Components locally (DC PS.3) Replication of instances of functional components in the cloud (DC PS.4)
Prioritize Processing	Functional component offer services for different priorities (DC PS.5)	Information holder for priority information necessary (DC PS.6)	Provide instances of different functional components for different priorities (DC PS.7) Priority-aware functional components with priority based processing and networking (DC PS.8)
Partition and parallelize	Multi-thread/multiprogramming aware Functional components (DC PS.9)	Information flow needs to be parallelizable (DC PS.10)	Location-aware deployment of functional components (DC PS.11) Deployment of functional components need to be according to data flow (DC PS.12)
Reduce computational complexity	No functional component (DC PS.13) Functional component with reduced capabilities (DC PS.14)	No Impact	Less functional component deployed (DC PS.15)
Distribute processing over time	Design components to schedule processing (DC PS.16)	Information holder for deadline (DC PS.17)	No impact

(continued)

Table 6.11 (continued)

Tactic	Impact on views		
	Functional	Information	Deployment and operation
Minimize the use of shared resources	Design functional components to minimize use of shared resources (DC PS.18)	No impact	Minimize communication distances (DC PS.19) Deployment to minimize use of shared resources (DC PS.20)
Reuse resources and results	History aware functional components (DC PS.21)	Cache results which are likely to be reused (DC PS.22)	Storage of information locally (DC PS.23) Storage of information remotely (DC PS.24) Storage of information local and remotely (DC PS.25)
Scale up or scale out	Design functional Components in a replicable way (DC PS.26) Design function components so that they can use cloud support (DC PS.27)	No impact	Provision of further resources (DC PS.28) Use services in the cloud (DC P.29)
Degrade gracefully	Functional Components need to be able to restart (DC PS.29) Functional components with rollback functionality (DC PS.30)	Support of rollback points (DC PS.31)	Replication of components (DC PS.32) Redundancy of resources (DC PS.33)
Use asynchronous processing	Asynchronous-aware functional component (DC PS.35)	No impact	No impact

6.9.3 *Design Choices Addressing Performance and Scalability*

Performance and scalability are closely related. In the Internet of Things, with its anticipated billion or trillion nodes both performance and scalability will play a crucial role. In Sect. 8.3.1 the Performance and Scalability Perspective together with a set of tactics are presented. In the following we applied the tactics from the Performance and Scalability Perspective to our Design Choices. We furthermore evaluated their expected impact on the Functional, Information, and Deployment and Operation Views.

Not all tactics are explained in detail in this section. The tactic “Make Design compromises”, for example, was omitted, as being too general and as the whole idea of the design choices it to make compromises. Additionally, as performance is something that is very dependent on both architecture and implementation it is

highly advisable to run through the corresponding activities like creating a performance model or conduct practical testing with measurements. The full list of activities are listed in Sect. 8.3.

6.9.3.1 Replication

The functional components (DC PS.1) and the information (DC PS.2) stored can be replicated to increase performance and scalability (DC PS.3). Having a single functional component is often against good scalability. The availability of information depends on the availability of the IoT device. Having instances of functional components and information available remotely (for example, in the cloud) usually increases both scalability and performance (DC PS.4). Nonetheless, in this case one needs to be enough connectivity and bandwidth provided.

6.9.3.2 Prioritize Processing

To be able to prioritize processing the functional components needs to be aware that it might be required to prefer one type of processing over the other. Therefore, the information model needs to be able to provide information that indicates priorities of processes, for instance high, normal, or low.. In terms of deployment the prioritized processing can be done with the help of the network stack (DC PS.7) or there can be different functional components for the different priorities (DC PS.8).

6.9.3.3 Partition and Parallelize

Partition and Parallelize aims towards increase both scalability, as well as, performance by making the functional components aware of multi-threading/multi-programming (DC PS.9). Furthermore the information needs to be partitionable (reduce interdependencies between information) (DC PS.10). The deployment can help a lot in partitioning, as in IoT access to IoT services are often locally distributed. This can be done either location aware (DC PS.11), or based on a data-flow model (DC PS.12).

As an example, the Virtual Entity resolution could be location-oriented, where a resolution server (RS) is responsible for indexing all connected things in a certain geographical area, called indexing scope. A Catalogue server then creates the Catalogue Index of every RS' indexing scope. A resolution request is redirected towards the RS whose indexing scope intersects the search scope of the request. Large-scale IoT systems are expected to have multiple administrative domains that must be handled by a federated resolution infrastructure. Different domains interact with each other by the means of a central domain directory or domain catalogue. Another possibility would be a federated infrastructure, in which Virtual Entities are clustered based on similarity. Dedicated places are in charge of the IoT Services

they offer and provide their descriptions as part of a distributed resolution framework. The framework is scalable and fault tolerant because of distribution.

6.9.3.4 Reduce Computational Complexity

Whenever possible the system can reduce the computational complexity, thus leading to a simpler system which needs less time and often energy. As an example, instead of a complex intrusion detection system, there could either be no intrusion detection at all (DC PS.13) or a less complex security by design (DC PS.14), e.g. a protocol stack with built in threshold-based protection against too many session initiations.

6.9.3.5 Distribute Processing Over Time

To reduce the number of resources needed it is often possible to distribute some processing tasks over time, when their results are not immediately necessary (DC PS.16). In case of hard real-time constraints this might not be always possible, but many system do not need real-time at all, or do only have soft real-time constraints. Distributing processing over time can help preventing the system from scaling or reduce the use of remote (over the web) services.

6.9.3.6 Minimize Used of Shared Resources

In many IoT systems the most scare and most expensive resource is bandwidth, especially in wireless battery powered systems. It is necessary to design the functional components accordingly and especially plan the deployment to avoid bottlenecks on the devices/resources.

6.9.3.7 Reuse Resources and Results

To be able to reuse resources and results the functional components need to be aware of a history for reuse (DC PS.21). The information model needs be aware of such caching mechanisms (DC PS.22). In terms of deployment the history can either be stored locally (DC PS.23), remotely (DC PS.24) or a combination of both (DC PS.25).

If the information history is stored locally (DC PS.23) the information history is stored on the IoT device that has produced the information over time. History information needs to be secured in the same way as the present information to avoid information leaks. If constrained IoT devices are used, then the storage size of information history as well as the information processing performance is limited: Having a local storage place for history information on each IoT Device requires less device performance and less effort to secure the history, but the single

information host is against good scalability. The availability of information history depends on the availability of the IoT device hosting the history.

DC PS.24 describes the case, where the information history is not stored on the IoT Device that has produced the information, but on a different IoT Resource, to which the information is uploaded to. The additional history resource needs to be secured too with either the same S&P policies as the original IoT Resource or different policies. A history resource in the cloud can perform better than IoT devices; the replication of information allows load balancing between history and present information which contributes to better scalability. The Information history still exists when the respective IoT device becomes unavailable.

Furthermore it is possible to combine the two aforementioned approaches (DC PS.25): The information history is stored on the IoT device that has produced the information as well as on a different IoT Resource replicating the information. History information that exceeds the capabilities of the hosting IoT device capabilities can be offloaded to high performance devices. This design choice contributes to high scalability as well as higher performance since the remotely stored history information is a replication of the locally stored information. Replicating information is cheaper to achieve by the device than retrieving ‘fresh’ information for every replication.

6.9.3.8 Scale Up or Scale Out

Scale up and scale out is one of the traditional ways to ensure scalability. Scale up (also known as vertical scalability) means providing more resources on a single system (DC PS.26/DC PS.28), scale out (also known as horizontal scaling) means providing more computing power by adding resources. In IoT it is usually not that easy to scale up or to scale out. One obvious possibility is, of course, to use cloud support (DC PS.27/DC PS.29). Migration in sensor networks is possible to some extent as well in a heterogeneous network.

6.9.3.9 Degrade Gracefully

Degrade gracefully is a property of a system, which allows it to continue operating properly even in the event of failure in one or more components. The functional components need to be able to restart either completely (reset) or to rollback to a previous stable state. In case of hardware failures redundancy and replication allow to continue working even when a device/resource fails.

6.9.3.10 Use Asynchronous Processing

Asynchronous processing is usually intrinsic in IoT systems. All functional components should be prepared to do asynchronous calculations and synchronization needs to be planned accordingly.

Table 6.12 Tactics and corresponding design choices for trust

Tactic	Impact on views		Deployment and operation
	Functional	Information	
Harden root of trust	The security policy defines how the root of trust may be accessed. (DC T.1)	No impact	Integration of IoT-A trust and reputation component (DC T.2)
	Secure implementation for protecting a root-of-trust based on hardware implementation (DC T.3)	No impact	Integration of a physically unclonable function (PUF) (DC T.4)
Ensure high quality of data	Protects data integrity and freshness by using a secure network encryption protocol (DC T.5)	Improvement of content dimension and intellectual dimension (DC T.6)	Integration of a secure network encryption protocol (DC T.7)
Infrastructural trust and reputation agents	Collects user reputation scores and calculates service trust levels (DC T.8)	Service description should include relevant aspects for what concerns trust evaluation (DC T.9)	Integration of IoT-A trust and reputation (DC T.10)
	Web of Trust system to establish the authenticity of the binding between a public key and its owner. (DC T.11)	No impact	Decentralized trust model (DC T.12)
Provide high system integrity	Evaluation of trust based on reputation (DC T.13)	No impact	Integration of a reputation framework for high integrity sensor networks (RFSN) (DC T.14)
Avoid leap of faith	Utilizes one-way hash chain to provide effective and efficient authentication (DC T.15)	No impact	Usage of lightweight authentication protocol (DC T.16)

6.9.4 Design Choices Addressing Trust

In Sect. 8.3.3.1 the Trust Perspective together with a set of tactics is presented.

In Table 6.12 all tactics together with their Design Choices are listed. A detailed description for each tactic follows the table.

6.9.4.1 Harden Root of Trust

The root-of-trust is the core component upon which the trust policy is based. The notion of a root-of-trust exists at multiple abstraction levels in a system, and can be software (less secure) as well as hardware (higher security). As an example for hardware realisation is RFID. The tags can be used to support anti-counterfeiting by using a security protocol based on public key cryptography. In this case their root-of-trust is based on a *Physically Unclonable Device* (PUF) (Verbauwhede and Schaumont 2007).

6.9.4.2 Ensure High Quality of Data

Information quality is improved in the technical dimension (e.g. timeliness and sampling). The suite of security protocols (SPINS) guarantees that an attack does not affect the remainder nodes in the network and thus preserves data integrity and freshness. In the context of the Information view it can be stated that data containing information is improved in terms of content dimension (e.g. accuracy or completeness) and intellectual dimension (e.g. reputation and trust). To reach this level of security a secure network encryption protocol must be implemented (Perrig et al. 2002).

6.9.4.3 Infrastructural Trust and Reputation Agents

The tactic “Infrastructural Trust and Reputation Agents for scalability” describes the presence of a Trust and Reputation component FC (Sect. 7.7.1). This impacts the information view as a Service Description should include relevant aspects for what concerns trust evaluation (type of deployment, tamper-proof features of hosting devices, authentication and authorization algorithms, etc. In case of peripheral devices the security of the deployment should be evaluated and asserted in the subject description. Furthermore the web of trust concept to establish the authenticity of the binding between a public key and its owner can be established. Its decentralized trust model is an alternative to the centralized trust model of a *Public Key Infrastructure* (PKI), which relies exclusively on a certificate authority (or a hierarchy of such).

6.9.4.4 Provide High System Integrity

To provide high system integrity the integration of Reputation framework for high integrity sensor networks (RFSN) can be considered (Ganeriwal and Srivastava 2004). It is capable of evaluating trust based on reputation and to act accordingly. Furthermore second hand information (experiences of other parties, e.g. nodes) about devices can be considered. It might be augmented by a Trust management system which calculates Trust values as a function of availability and packet forwarding.

Table 6.13 Omitted tactics for the trust perspective

Tactic	Reason
Ensure physical security and implement tampering detection	Pervasive deployment of IoT devices makes such devices accessible to malicious users
Consider device security in the global system design	Devices that are not tamper-proof can be compromised. Although this aspect is related to the deployment view, it has impacts on the design of the overall system and trust evaluation
Consider the impact of security/performance trade-offs on trust	This must be evaluated for each use case during the design phase by means of tests such as simulation. For that reason, no DC can be proposed
Use security imprinting	Out of scope for IoT-A since devices are not covered in the IoT Reference Architecture
Balance privacy vs. non-repudiation (accountability)	If system requirements include non-repudiation, these will necessarily impact the privacy feature of the designed system. Privacy can be granted by using an Identity Management. This component, run by a third party is trusted for what concerns both privacy protection and ability to track back malicious actions

6.9.4.5 Avoid Leap of Faith

The avoidance of leap of faith increases the overall security; however, it might limit the communication between certain parties as strong authentication is not feasible in each case (e.g. constrained devices). From a functional point of view one option can be a one-way hash chain to provide effective and efficient authentication. This feature can be implemented by using a Lightweight Authentication protocol (Lu and Pooch 2005).

For most of the tactics a design choice proposal is given, however for different reasons it is not possible to provide appropriate design choices for all tactics. The tactics not considered are presented in Table 6.13 with reasons for the omission.

6.9.5 Design Choices Addressing Security

In Sect. 8.3.3.2 the Security Perspective together with a set of tactics is presented. The Design Choices addressing security are presented in Table 6.14 showing the impact on architectural views by applying tactics relevant for security concerns.

6.9.5.1 Subject Authentication

For subject authentication two options are presented here. The first is the authentication over an encrypted channel while the other one is a crypto-based authentication solution over an open channel. The former uses the IoT-A Authentication FC (Sect. 7.7.2) while for the latter a peer-to-peer communication is realised over an insecure channel.

Table 6.14 Tactics and corresponding design choices for security

Impact on views	
Tactic	Deployment and operation
Subject authentication	Integration of IoT-A authentication FC (DC S.2)
Use access policies	Peer-to-peer authenticated communications over an insecure channel must be possible (DC S.4) IoT-A Authorisation FC component (DC S.7)
Secure communication infrastructure	No impact
Secure peripheral networks (link layer security, secure routing)	IoT-A end to end communication FC, Network communication FC and key exchange and management FC (DC S.12)
	IoT-A hop to hop communication FC, network communication FC and key exchange and management FC (DC S.15) (Sect. 7.7.2)
	End-to-end security protocol to ensure wireless communication security (DC S.18)
	Integration of secure routing protocols in the Network Communication component (DC S.20)
Functional	Information
Authentication over encrypted channel (DC S.1)	No impact
Crypto-based authentication over open channel (DC S.3)	No impact
Policy-based service access (DC S.5)	Stored information must be managed in a way to support access control mechanisms (DC S.6)
Unrestricted access to service (DC S.8)	Stored information is not protected (DC S.9)
End-to-end encryption (DC S.10)	Information transmission channel between device and application is secured (DC S.11)
Hop-to-hop encryption (DC S.13)	Information transmission channel between device and application is secured (DC S.14)
Cryptographic protocols ensuring confidentiality, integrity, authentication of subjects (DC S.16)	Communication channel between two subjects is secured (DC S.17)
Link-layer encryption and authentication, multipath routing (DC S.19)	No impact

6.9.5.2 Use Access Policies

The tactic of using access policies is a crucial aspect in IoT. Two main functional principles can be distinguished. The policy-based service access uses access control mechanisms to manage to access to information. Therefore the information must be managed accordingly so that it supports the used mechanism. This option can be realised by using the IoT-A Authorisation FC component (Sect. 7.7.2). The other possibility is to grant unrestricted access to services. This should be only done in those cases in which data security is not relevant.

6.9.5.3 Secure Communication Infrastructure

Securing the communication infrastructure focuses on delivering a secure and robust environment for the transmission of critical data. This can be obtained by using end-to-end or hop-to-hop encryption. In both cases the information transmission channel in which the information flow from a device to an application through an IoT service happens is completely secured. The end-to-end encryption uses therefore the IoT-A End to End Communication FC and Key Exchange and Management FC. Furthermore the Network Communication FC, which takes care of enabling communication between networks through Locators (addressing) and ID Resolution, is necessary (Sect. 7.7.2). For the hop-to-hop encryption the only difference is the usage of the IoT-A hop-to-hop Communication FC. For wireless communication security the implementation of an end-to-end security protocol which ensures confidentiality, integrity and authentication of subjects can also be considered (Perrig et al. 2004).

6.9.5.4 Secure Peripheral Networks (Link Layer Security, Secure Routing)

To secure peripheral networks a link-layer encryption and authentication combined with a multipath routing can be considered. This requires the integration of secure routing protocols in the Network Communication component (Karlof and Wagner 2003).

For most of the tactics a design choice proposal is given, however for different reasons it is not possible to provide appropriate design choices for all tactics. The tactics not considered are presented in Table 6.15 with reasons for the omission.

6.9.6 *Design Choices Addressing Privacy*

In Sect. 8.3.3.3 the Security Perspective together with a set of tactics is presented. The Design Choices addressing Privacy are presented in Table 6.16 showing the impact on architectural views by applying tactics relevant for Privacy concerns.

Table 6.15 Omitted tactics for the Security Perspective

Tactic	Reason
Harden infrastructural functional components	Infrastructural functional components are critical components that can compromise the whole system if compromised
Avoid wherever possible wireless communication	Wireless communication generally uses a shared medium for communication which, in turn, allows easy interception of link layer communication
Physically protect peripheral devices	Pervasive deployment of IoT devices makes such devices accessible to malicious users. While how to protect these devices is outside the scope of the IoT Reference Architecture (devices not covered!), this vulnerability must be taken into account in secure designs
Avoid OTA device management	No DC proposal possible as most of the devices connected in IoT must be managed over the air if at all possible

Table 6.16 Tactics and corresponding design choices for privacy

Tactic	Impact on views		Deployment and operation
	Functional	Information	
Pseudonymisation	Creation of a fictional identity (root identity, secondary identity, pseudonym or group identity) (DC P.1)	No impact	Integration of IoT-A identity management FC (DC P.2)
Avoid transmitting identifiers in clear	Encryption mechanisms for wireless connections (DC P.3)	No impact	Integration of a wireless security algorithm (DC P.4)
Minimize unauthorized access to implicit information	Access control management (DC P.5)	Stored Information must be managed in a way to support access control mechanisms (DC P.6)	IoT-A authorisation FC (DC P.7)
	Enablement of a scalable and secure key distribution between communicating subjects (DC P.8)	No impact	Encrypt communication with Resolution Components and with Services (e.g. KEM FC) (DC P.9)
Enable the user to control the privacy settings	Addresses privacy questions so that a user can operate anonymously (DC P.10)	No impact	IoT-A identity management FC (DC P.11)
Privacy-aware identification	Authentication of the responding host, the initiating host can stay anonymous (DC P.12)	No impact	Requires TLS and DTLS support (DC P.13)

6.9.6.1 Pseudonymisation

The tactic “Pseudonymisation” refers to a procedure by which fields that enable identification of a user within a data record or subject are replaced by one or more artificial identifiers. The purpose is to render the subject less identifiable and this way lower IoT user (e.g. customer or patient) objections to its use. This is functionally implemented by the creation of a fictional identity (e.g. root identity, secondary identity, pseudonym, or group identity) and can be realised by integrating the IoT-A Identity Management FC (Sect. 7.7.3).

6.9.6.2 Avoid Transmitting Identifiers in Clear

The transmission of identifiers in clear should be avoided in general. In a WSN, a base station is not only in charge of collecting and analysing data, but also used as the gateway connecting the WSN with outside wireless or wired network. In order to have a defence against local adversaries, the location information or identifier of the base station is sent in clear in many protocols. This information must be hidden from an eavesdropper, which can be done by traditional cryptographic techniques (encryption). One option for encrypting wireless connections is the integration of a wireless security algorithm proposed by (Peris-Lopez et al. 2007).

6.9.6.3 Minimize Unauthorized Access to Implicit Information

Unauthorized access to implicit information (e.g. deriving location information from service access requests) must be restricted at all events. Access control management as well as the enablement of a scalable and secure key distribution between communication subjects can be considered to achieve this objective. In the former case the information stored must be managed in a way so that the access control mechanism is supported. For deployment of this function the IoT-A Authorisation FC can be considered. For the secure key distribution the resolution components should be augmented by a Key Exchange Management component such as the one from IoT-A.

6.9.6.4 Enable the User to Control the Privacy Settings

Users should be given the opportunity to control their privacy settings. Hence, one option is the control of acting anonymously. This function can be realised by integrating the IoT-A Identity Management FC which creates a fictional identity (root identity, secondary identity, pseudonym, or group identity) along with the related security credentials for users and services to use during the authentication process.

Table 6.17 Omitted tactics for the privacy perspective

Tactic	Reason
Validate against requirements	Too general, no DC proposal possible
Consider the impact of security/performance trade-offs on privacy	This must be evaluated for each use case during the design phase. For that reason, no DC can be proposed
Balance privacy vs. non-repudiation (accountability)	This must be evaluated for each use case during the design phase. For that reason, no DC can be proposed

6.9.6.5 Privacy-Aware Identification

In human-to-thing and thing-to-thing interactions, privacy-aware identifiers might be used to prevent unauthorized user tracking. Similarly, authentication can be used to prove membership of a group without revealing unnecessary information about an individual. *Transport Layer Security* (TLS) and *Datagram Transport Layer Security* (DTLS) provide the option of only authenticating the responding host. This way, the initiating host can stay anonymous (Heer et al. 2011).

For most of the tactics a design choice proposal is given, however for different reasons it is not possible to provide appropriate design choices for all tactics. The tactics not considered are presented in Table 6.17 with reasons for the omission:

6.9.7 Design Choices Addressing Availability and Resilience

The Chapter in this document concerned with the Availability and Resilience Perspective (Sect. 8.3.4) lists tactics addressing the desired quality of the system to be designed as shown in Table 6.18.

In this Section design choices are presented that apply most of the tactics listed in Table 6.19. The tactics not considered here are given at the end of this Section with an explanation why they have been omitted. Table 6.19 presents for each tactic one or more architectural design choices together with their impact on the architectural views introduced in Chap. 8.

6.9.7.1 Use High Availability Clustering

For design choice ‘VE Resolution location-oriented (DC A.1)’ a resolution server (RS) is responsible for indexing all connected things in a certain geographical area, called indexing scope. A Catalogue server then creates the Catalogue Index of every RS’ indexing scope. A resolution request is redirected towards the RS whose indexing scope intersects the search scope of the request. Large-scale IoT systems are expected to have multiple administrative domains that must be handled by a federated resolution infrastructure. Different domains interact with each other by the means of a central domain directory or domain catalogue. Communication between framework domains needs to be secured. The framework performs faster through a divided search space. Indexing scope can be adjusted according to usage load. The framework scales by adding more RSs. With this approach it is

Table 6.18 Tactics addressing Availability and Resilience

Desired quality	The ability of the system to be fully or partly operational as and when required and to effectively handle failures that could affect system availability
Tactics	Select fault-tolerant hardware Use high-availability clustering and load balancing Log transactions Apply software availability solutions Select or create fault-tolerant software Design for failure Allow for component replication Relax transactional consistency Identify backup and disaster recovery solution

impossible to retrieve things based on identifiers. Fault tolerance is achieved through data distribution and index data replication. The central domain directory is potential single point of failure. There is no theoretical limit on indexed things, but indexing scope is bound to geographic location (De 2012).

In design choice ‘VE Resolution domain-oriented (DC A.2)’ a domain-oriented VE Resolution approach organises the resolution framework in hierarchically organised domains similar to Domain Name System (DNS). The hierarchy is built according to the hierarchy of things captured by Virtual Entities from higher granularity to lower granularity, e.g. country → city → district → building → room. The resolution framework performs faster than an unclustered resolution solution through divided search space; its complexity is of $O(\log n)$ in best case, and $O(n)$ in worst case, where n is the number of VEs hosted by the resolution framework. Load balancing is supported through replication, and a Resource can be member of different domains at a time. Fault tolerance is supported through distribution and redundancy; the framework evolves with the number of things connected (De 2012).

For design choice ‘VE Resolution Semantic Web-oriented (DC A.3)’ Semantic Web technologies are used to annotate Virtual Entity descriptions in a way machines can interpret them. This overcomes the need for exact syntactic match-making between resolution request and search terms in the resolution infrastructure. The search space of the resolution infrastructure is indexed by an unsupervised machine-learning technique and clustered through latent factors derived from the learning. This design is independent from the deployment of the resolution infrastructure. Distribution and replication is supported by this approach, but depends on implementation on how it is done. Semantic interoperability is achieved through shared ontologies, after extending ontologies the training model needs to be updated (De 2012).

A peer-to-peer infrastructure will maintain no centralised servers in design choice ‘VE Resolution Peer-to-Peer-oriented (DC A.4)’, all data is distributed in the network along with sophisticated retrieval and routing mechanisms. There are several approaches on how to distribute the data (pure, centralised indexing server, distributed hash tables). The latter approach is the recommended one for IoT

Table 6.19 Design Choices addressing availability and resilience

Tactic	Impact on views		
	Functional	Information	
Use high availability clustering	VE resolution location-oriented (DC A.1) (De 2012)	Information model requires data type for defining scope for location of interest	VE resolution instances for each location cluster
	VE resolution domain-oriented (DC A.2) (De 2012)	Information model needs data type for defining types of resources	Resolution framework is organised hierarchically
	VE resolution semantic web-oriented (DC A.3) (De 2012)	Information model needs to be encoded according to semantic web standards	Search space of resolution framework needs to be indexed by certain machine-learning technique
	VE resolution peer-to-peer-oriented (DC A.4) (De 2012)	No impact	No centralised server needed
Use load balancing	Requires function that monitors load of components and triggers load balancing algorithm	Component descriptions need metric to measure current work load and defined load limits	Scaling out approach additional clones of components need to be available (DC A.5)
Log transactions	Apply circular logging strategy (IBM 2012) (DC A.6)	Information model needs concepts for transactions with unique identifiers	Storage for transaction logs needed
Design for failure	Apply archive logging strategy (IBM 2012) (DC A.7)	Like above, but transactions need to be marked as either active or inactive additionally	External storage needed for large logs
	Provide functionality to reserve spare resources and replace failed ones (Newelligence 2012) (DC A.8)	No impact	Spare resources are kept on hold until an operating resource needs to be replaced, requires higher amount of resources
	Prefer service choreography FC over service orchestration FC (Sect. 8.2.2.3)	Identifiers in information model need to be unique for clones of FCs and across distributed system	No single FC or centralised FC (DC A.9)
	Provide FC that monitors latency (DC A.10)	Provide means to specify latency and timeout limits	No impact

<p>Allow for component replication</p>	<p>Provide FC that implements state-machine (active) replication (Wikipedia 2013d) (DC A.11)</p> <p>Provide FC that implement transactional replication (Microsoft 2013) (DC A.12)</p> <p>Provide FC that implements virtual synchrony (Wikipedia 2013e) (DC A.13)</p> <p>Requires conflict resolution functionality</p>	<p>FC needs to be modelled as state-machine</p> <p>Also incremental changes of information can be replicated, leads to inconsistency, though, if not completed</p> <p>Make replicated information indistinguishable non-replicated information</p> <p>No impact</p>	<p>To support F failures you must have at least 2F+1 replicas of the component</p> <p>Good performance, near real-time replication possible</p> <p>Very high performance</p>
<p>Relax transactional consistency (DC A.14)</p>	<p>Requires conflict resolution functionality</p>	<p>No impact</p>	<p>Needs more resources through replication</p>
<p>Identify backup and disaster recovery strategy</p>	<p>Preventive measures (DC A.15)</p> <p>Detective measures (DC A.16)</p> <p>Requires data-replication functionality</p> <p>Requires monitoring of indicators for disastrous events</p>	<p>Requires consistency among replicated data</p> <p>Requires modelling of disastrous events to be looked for and propagation of those</p> <p>Requires storage of configuration history</p>	<p>Replicated and archived data are stored off-site in the cloud</p> <p>Disaster monitoring needs to be operated independently of components to be monitored</p> <p>Disaster recovery needs to be operated independently of components to be recovered</p>
<p>Corrective measures (DC A.17)</p>	<p>Requires restoring of previously back-upped configurations</p>	<p>Requires storage of configuration history</p>	<p>Disaster recovery needs to be operated independently of components to be recovered</p>

Resolution infrastructures. Resolution requests result in traffic complexity of $O(n)$ in worst case and $O(\log n)$ in best case, where n is the number of VEs managed by the resolution framework. The framework is stable and robust through distribution and redundancy (De 2012).

6.9.7.2 Load Balancing

The ‘Scale out approach (DC A.5)’ monitors the load of FCs during runtime and triggers offloading tasks to another less busy instance of the respective FC to avoid the FC being overloaded and therefore becoming a performance bottleneck or even out of function. The decision at what limit an FC is considered to be critically busy and to trigger off-loading to another instance is application specific, but the information model needs to provide some metric to specify those parameters for FCs.

Logging Transactions

‘Circular Logging (DC A.6)’ is a strategy that leads to overwriting old data when designated size of log is reached (IBM 2012). This approach does not support incremental backup strategy. Transactions need to be logged with unique id and status of their completion, indicating which functions need redoing and which need undoing. Apply this Design Choice if storage space for logs is restricted. This strategy provides better performance compared to archive logging.

‘Archive Logging (DC A.7)’ keeps a complete archive of all transactions (IBM 2012). Recent transactions need to be flagged as active, older transactions as inactive. The archived logs grow over time so that external storage is needed on constraint devices. This strategy adds functionality for retrieving the external archive also for rollback and restore.

Design for Failure

The overall tactic can be further divided into more specific tactics that are presented as design choices here. The first sub-tactic is ‘Acquiring more resources than needed and replace failed ones (DC A.8)’. By applying this tactic more resources are allocated for task execution than normally required. Besides allocating the resources essentially necessary spare resources are reserved that could execute the same task as the essential ones but are kept on hold. This is a precaution in case a resource fails during runtime and a spare resource can take over the task of the one that failed. Resource in this sense includes all computational resources, network resources and IoT Resources, meaning all FCs in the ARM. A typical FG that implements resource reservation is Service Organisation that is responsible for allocating IoT Services to service requests (see Sect. 8.2.2.3). Applying this tactic requires a higher number of resources essentially required.

Another approach is to aim at having ‘No FC or centralised FCs (DC A.9)’. The goal is to develop designs that avoid single points of failure, like centralised FCs or

FCs with just one instance. If a single FC fails no other instance was able to replace its functionality. By applying this tactic more than one instance of FCs are provided by the system so that their functionality can still be assured in case one instance becomes unavailable. For Service Organisation FG the decentralised Service Choreography FC can be preferred over Service Orchestration which requires a central orchestration engine (see Sect. 8.2.2.3). The decentralised choreography approach reduces the risk for a single point of failure.

To apply the design choice ‘Treat Long Latency as potential failure (DC A.10)’ the system design provides an FC that treats any long latency as a potential failure. For instance the round-trip-time for request-response-protocols is measured and a deadline is set as acceptable. After the deadline has passed the system treats the behaviour as potential failure and reacts in an appropriate manner, e.g., by querying another instance of the same FC.

Allowing Component Replication

The design choice ‘State-machine (active) replication (DC A.11)’ allows detection of faults by replicating service requests and comparing the service results to each other. If all results are identical no fault is assumed, if they are different it still needs to be analysed which of the results is faulty and which is correct (Wikipedia 2013d). To apply this technique some replication functionality needs to be implemented that multiplies the request to different instances of FCs. To assure fault detection $2F+1$ replicas of the tested FC need to be held where F is the number of faults to be detected. The fault detection algorithm requires the tested FC to be modelled as state-machine.

‘Transactional replication (DC A.12)’ is used in server-to-server environments typically, in which incremental information changes need to be propagated to subscribers in nearly real-time (Microsoft 2013).

The choice ‘Virtual synchrony (DC A.13)’ is especially suitable for systems in which information evolves extremely rapidly. Applications are executed in process groups and the processes within the group update each other about execution progress by sending state updates. Implementing this technique requires functionality to join process groups, register event handler and send multicasts to group members. Consistency among information replicas can be achieved easily, thus virtual synchrony is suitable for systems with high evolution of information (Wikipedia 2013e).

Relaxing Transactional Consistency

To follow this tactic the ‘BASE architecture (DC A.14)’ can be applied. The ‘BASE (Basically Available, Soft-state, Eventually consistent) architecture’ is applicable in systems supporting distributed transactions with optimistic replication strategy. In this approach replicas of information are sent through a distributed system via transactions and ‘eventual consistency’ among the replicas is achieved by either the update reaches the replica or the replica retires from service (Wikipedia 2013f). BASE requires some conflict resolution functionality and additional system

resources in order to find failure in transactions. The approach is applicable for high performance designs.

Backup and Disaster Recovery Strategy

The following design choices should not be seen as alternative choices to apply one tactic; the three choices are rather three controls that can help to specify a disaster recovery plan for the system to be designed (Georgetown 2013). Therefore all three choices can be applied alongside.

The choice ‘Preventive measures (DC A.15)’ is aimed at preventing disastrous events, like data-loss, from occurring. To achieve this data is replicated to have identical copies in reserve in case the original data gets lost. Consistency among the data replicas needs to be assured by the design. To minimise risks the replicas are better stored at different locations than the original data, preferably in the cloud.

‘Detective measures (DC A.16)’ aim at detecting or discovering unwanted events by monitoring indicators for unwanted events, like measured values that exceed a certain range. This strategy requires an Information Model of those unwanted events together with their indicators that are used to detect the unwanted event. The event detection should be operated independent of the subsystem that is monitored to make sure the unwanted events can be detected.

The design choice ‘Corrective measures (DC A.17)’ is aimed at correcting or restoring systems after disastrous events have occurred. Assuming the previous two choices have been implemented, meaning the preventive methods have been applied and the disastrous event has been detected correctly, the system can be restored to working order again. Backups of system configurations that have worked correctly before are restored. A configuration history (Sect. 8.2.2.8) provides the functionality needed for restoring working configurations. The system correction process needs to be operated independently of the system to be restored.

Some of the tactics listed in Sect. 8.3 are not considered here because they are too specific to particular implementations:

- Select fault-tolerant hardware;
- Apply software availability solutions;
- Select or create fault-tolerant software.

6.9.8 Design Choices Conclusion

This section has presented design choices for architects who are driven by requirements for system quality capabilities like performance and scalability, evolution and interoperability, availability and resilience as well as aspects concerning trust, security, and privacy. An architect is guided by the presented design choices in supporting the targeted system quality attributes. In cases where the recommended design choice is one developed during the IoT-A project a reference is given where an architect can find more detailed information about the respective design choice.

The design choices listed in this section are as generic as possible addressing capabilities that are agnostic of particular functional requirements. The architect is still left with the choice which system capabilities are the most important ones for the system to be specified. In general trade-offs need to be made between for instance security and performance since security always involves more data and communication overhead that needs to be processed.

The optimal selection of design choices is dependent on the actual use case and therefore a one-fits-all complete solution cannot be given in this section. It rather needs to be made by architects according to their functional requirements which are not known in the context of this document.

What this document can provide instead is an example for a concrete architecture that is designed according to a sample use case. Architects shall find useful hints for applying the ARM to concrete architectures including a selection of appropriate design choices presents in this section. The sample concrete architecture is described in Chap. 11.

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