Chapter 3
The IoT Architectural Reference Model as Enabler

Martin Bauer and Joachim W. Walewski

As identified in the previous chapter, IoT-A has created an "Architectural Reference Model" (IoT ARM) as the common ground for the Internet of Things. The core idea is that the IoT ARM provides a common structure and guidelines for dealing with core aspects of developing, using and analysing IoT systems. The first part of this chapter provides a non-exclusive list of the beneficial uses of the IoT ARM. In the second part we focus on the role of the IoT ARM in the architecture development process.

3.1 Using the IoT ARM

In the following we present a non-exclusive list of the beneficial uses of the IoT ARM. The order in which they are discussed does not imply any ranking – we list them according to their degree of abstraction and remoteness from the product: i.e. the first usage type is concerned more with generic enabling (abstract and remote), while the last usage type concerns how the IoT ARM can be used for procuring system solutions (concrete, close to business). The usage type that is more important to any specific use of the IoT ARM depends on the perspective of the actors involved. A manager of an IoT development process, for instance, is more likely to favour the enabling aspects of the IoT ARM, while a procurement department is more likely to favour concrete advantages that are closer to the business process itself.

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

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

A. Bassi et al. (eds.), *Enabling Things to Talk*, DOI 10.1007/978-3-642-40403-0_3,

3.1.1 Cognitive Aid

When it comes to product development and other activities, an architectural reference model is of fourfold use.

Firstly, it helps to guide discussions, since it provides a language everyone involved can use, and which is intimately linked to the architecture, the system, the usage domain, etc.

Secondly, the high-level view provided in such a model is of high educational value, since it provides an abstract but also rich view of the domain. Such a view can help people new to the field to "find their way" and to understand the special features and intricacies of IoT.

Thirdly, the IoT ARM can assist IoT project leaders in planning the work at hand and the teams needed. For instance, the Functionality Groups identified in the IoT Functional View of the IoT system can also be understood as a list of independent teams working on an IoT system implementation. The Process Chapter (Chap. 6) provides more insight on how the IoT ARM can support the architecture generation process and also about how to separate it into different activity "islands". This type of approach is particularly interesting for enterprise architecture frameworks that incorporate system-architecting processes. Typically, these enterprise frameworks provide institutional rules and prescriptions for how the system-architecting process is to be conducted. The IoT ARM can inform such institutional rules and prescriptions. An example of the latter is the Zachman framework (Zachman 1987).

Fourthly, the IoT ARM helps to identify independent building blocks for IoT systems. This constitutes very valuable information when dealing with questions such as system modularity, processor architectures, third-vendor options, re-use of components already developed, etc.

3.1.2 Reference Model as a Common Ground

Establishing a common ground for a field is not an easy task. In order to be effective, it has to capture as many pertinent vantage points as possible. Establishing the common ground for the IoT encompasses defining IoT entities and describing their basic interactions and relationships with each other. The IoT ARM provides exactly such a common ground for the IoT field.

3.1.3 Generating Architectures

One of the main benefits is the use of the IoT ARM for generating compliant architectures for specific systems. This is done by providing best practices and

guidance for translating the IoT ARM into concrete architectures. For an overview on this, see Chap. 5. The benefit of this type of generation scheme for IoT architectures is not only a certain degree of automation in this process, and thus lower R&D efforts, but also that the decisions made follow a clear, documented pattern as described in Chap. 6.

3.1.4 Identifying Differences in Derived Architectures

When using the aforementioned IoT ARM-based architecture process, any differences in the derived architectures can be attributed to the special features of the use case in question and the design choices related to this case (Shames and Yamada 2004). When applying the IoT ARM, a list of system function blocks, data models, etc., together with predictions of system complexity, etc. can be derived for the architecture generated. Furthermore, the IoT ARM defines a set of tactics and design choices for meeting qualitative system requirements (for more details, see Chap. 6, Design choices). All of these facts can be used to predict whether two derived architectures will differ and where they will do so.

The IoT ARM can also be used for reverse mapping. System architectures can be cast in the "IoT ARM" language and the resulting "translation" of the system architectures is then stripped of incompatible language and system partitions and mappings. The differences that remain are then true differences in architecture.

3.1.5 Achieving Interoperability

As we explain later on in this book (see Chap. 6 on design choices), fulfilling qualitative requirements through the architecting process inevitably leads to design challenges. Since there is usually more than one solution to each of the design challenges (we refer to these solutions as design choices), the IoT ARM cannot guarantee interoperability between any two concrete architectures, even if they have been derived from the same requirement set. Nevertheless, it is an important tool in helping to achieve interoperability between IoT systems. This is facilitated by the design-choice process itself. During this process, one identifies and tallies the design choices made. By comparing the design choices made when deriving two architectures, one can readily identify where in the architecture measures are necessary to achieve interoperability. Interoperability may be achieved a posteriori by integrating one IoT system as subsystem in another system, or by building a bridge through which key functionalities of the respective other IoT system can be used. Notice though that these workarounds often fall short of achieving full interoperability. Nevertheless, building bridges between such systems is typically much more straightforward than completely re-designing either system and usually fair interoperability can be achieved.

3.1.6 System Roadmaps and Product Life Cycles

Above we discussed how the design choices made in order to derive a particular architecture, and also the features selected, are instrumental in describing the difference between two architectures. As well as identifying the differences between two "foreign" architectures, this approach can also be used to map the evolution of architectures. For instance, design choices are tied to qualitative requirements. Let us assume that during the requirements process (see Chap. 6, Sect. 6.4), two disjoint "design choice" islands are identified, i.e. groups of design choices that lead to non-interdependent functionalities, data models, etc. In this case, it is possible to embody only one "design choice" island in the systems produced and to embody the full set of design choices in the next product generation. Thus, the IoT ARM can be used to devise system roadmaps that lead to minimum changes between two product generations while still guaranteeing a noticeable enhancement in system capability and features. This approach also helps the designer to formulate clear and standardised, requirements-based rationales for the system roadmap chosen and the product life cycles that result from the system roadmap.

3.1.7 Benchmarking

Another important use of the IoT ARM is benchmarking. For example, NASA used a reference architecture that described its envisaged exploration vehicle in order to receive better benchmarking tenders during a public bidding process for the said exploration vehicle (Tamblyn et al. 2007). While the reference model prescribed the language to be used in the systems/architectures to be assessed, the reference architecture stated the minimum (functional) requirements for the systems/architectures. By standardising the description and also the ordering and delineation of system components and aspects, this approach also provided the benchmarking process with a high level of transparency and inherent comparability. Using this approach, besides just "ticking" off the minimum features each tender has to fulfil, even more insight can be gained into the proposed system. For instance, the number and "richness" of functional components belonging to the system and their interaction patterns allow an appreciation of the system complexity both in terms of composition and structure but also in terms of interaction. This information can be gleaned from the IoT Functional View (functional decomposition, interactions), the IoT Information View (data flow, data complexity) and the IoT Deployment View. It makes judging the overall system complexity easier during the tender review phase.

3.2 Architecture Development Process Based on the IoT ARM

Following the overview of the different cases of usage in which the IoT ARM plays a beneficial role, we will now focus on how the IoT ARM can be used during the process of generating concrete IoT architectures suitable for specific applications and use cases. We will first discuss the idea behind reference models and reference architectures and the underlying methodology.

The process of developing an architecture is about finding a solution to a pre-defined goal. In turn, the development and description of architectures is a modelling exercise. It is important to point out that the modelling itself does not take place in a vacuum but is based on a thorough understanding of the domain to be modelled. In other words, any architecture development is contingent on the understanding of the domain in question. The same is true for a generalisation of this process, i.e. the derivation of reference architectures. Thus, reference architectures, such as the one presented in this book, also have to be based on a detailed understanding of the domain in question. This understanding is commonly provided in the form of a reference model.

3.2.1 Reference Model and Reference Architecture

Reference models and reference architectures provide a description that is more abstract than what is inherent to actual systems and applications. They are more abstract than concrete architectures that have been designed for a particular application with particular constraints and choices. From literature, we can extrapolate the dependencies between a reference architecture, architectures and actual systems (see Fig. 3.1) (Muller 2008). Architectures do help in designing, engineering, building and testing actual systems. At the same time, a better understanding of system constraints can provide input for the architecture design, and this allows future opportunities to be identified. The structure of the architecture can be made explicit through an architecture description, or it is implicit through the system itself. Extracting essential components of existing architectures, such as mechanisms or the use of standards, allows the definition of a reference architecture.

Guidelines can be linked to a reference architecture in order to derive concrete architectures from the reference architecture (Fig. 3.2, left). These general architecture dependencies apply to the modelling of the IoT domain as well.

The transformation step from an application-independent model to a platform-independent model is informed by guidelines. The step from platform-independent model to platform-specific model is discussed later in this chapter.

While the model presented in Fig. 3.1 stops at the reference architecture, the IoT-A Architectural Reference Model goes one step beyond this and also defines a

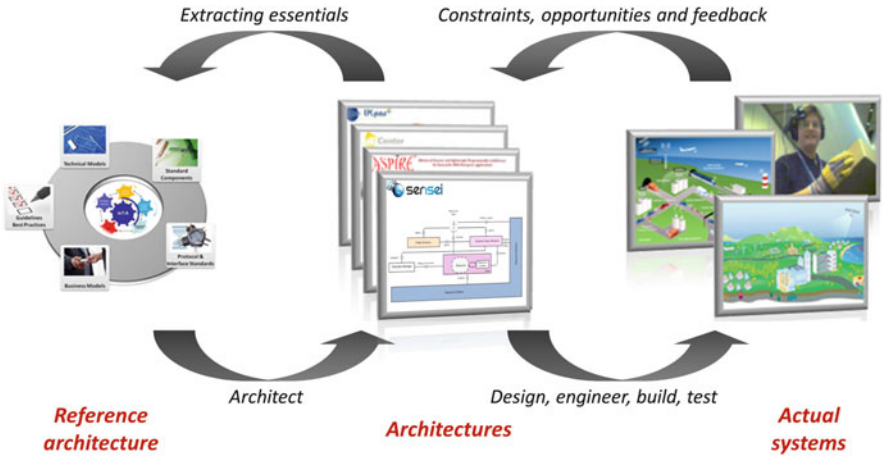

Fig. 3.1 Relationship between a reference architecture, architectures and actual systems (Adapted from Muller (2008))

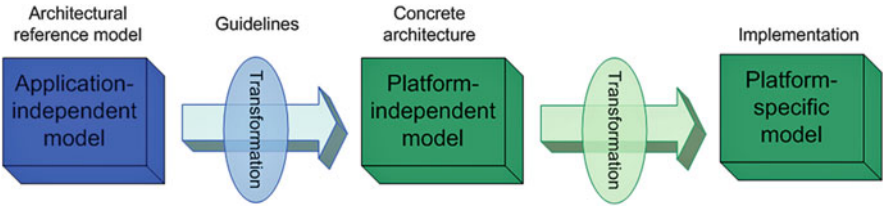

Fig. 3.2 Derivation of implementations (platform-specific models) from an architectural reference model (application-independent model) via the intermediate step of a concrete architecture (platform-independent model)

reference model. As already discussed, a reference model provides the ground for a common understanding of the IoT domain by modelling its concepts and their relationships. A detailed description of the IoT Reference Model can be found in Chap. 7.

3.2.2 Generating Architectures

Now that we have a general understanding about reference models, reference architectures and their relationships, the important question is how to derive the appropriate concrete architecture from the reference architecture. We dedicate an entire chapter to this issue, namely the Process Chap. 6, where we describe all aspects in great detail. However, the reader needs at least some appreciation of the

role of the IoT ARM here in order to take full advantage of the IoT Reference Model and the IoT Reference Architecture in the chapters in-between.

When applying the IoT ARM in designing systems, it is likely that in each individual case, different architectures will result. Thus, while Fig. 3.2 gives the impression that the process of translating the reference architecture into a concrete architecture is independent of the use case itself, this is, in reality, not so – the guidelines and the engineering practices chosen rely on a use case description and the requirements. This fact is reflected in Fig. 3.3. The role of the IoT ARM is to provide transformation rules for translating the rather abstract models into a concrete architecture. This step is strongly influenced by the use case and the related requirements. One entry point for this information is during the process of design choices, i.e. when the architect favours one avenue for realising the functionality or quality needed over another. The IoT ARM also recommends design patterns for such choices. The IoT ARM does not operate in a design vacuum but should be applied together with proven design process practices, which in themselves are contingent upon the guidelines provided and upon the use case and the requirements.

In Chaps. 7 and 8 we describe how both the IoT Reference Model and the IoT Reference Architecture can be used in this design process. Even though we describe the design process in a linear fashion, remember that in practice this will not always be the case. Depending on the engineering strategies used, some of the steps can be done in parallel or may even have to be reiterated due to additional understanding gained during the process or due to changes in the requirements.

3.2.3 Choice of Design and Development Methodology

The choice of a design and development methodology can be understood in two ways: firstly, a methodology for the IoT ARM development and secondly, a methodology for the generation of specific concrete architectures. We have so far only provided high-level views of either case. In reality, more guidance is required, i.e. a recipe for how to derive all aspects of the IoT ARM model as well as how to derive guidelines for the application of the IoT ARM for the generation of architectures.

In the case of the IoT ARM there are, to our knowledge, no standardised approaches for developing such a model. Furthermore, compared to typical reference architecture domains, the IoT usage domain is extremely wide and varied, and common denominators are thus rather few and abstract. For examples of reference architectures and models, the reader is directed to the following literature: (Consultative Committee 2006; MacKenzie et al. 2006; Muller 2008; Open GeoSpatial Consortium 2002; Shames and Yamada 2004; Tamblyn et al. 2007; Usländer 2007). This high level of abstraction in terms of the domain to be modelled stands in contrast to the input needed for established and standardised methodologies such as Aspect-Oriented Programming (AOP), Model-Driven Engineering (MDE), Pattern-Based

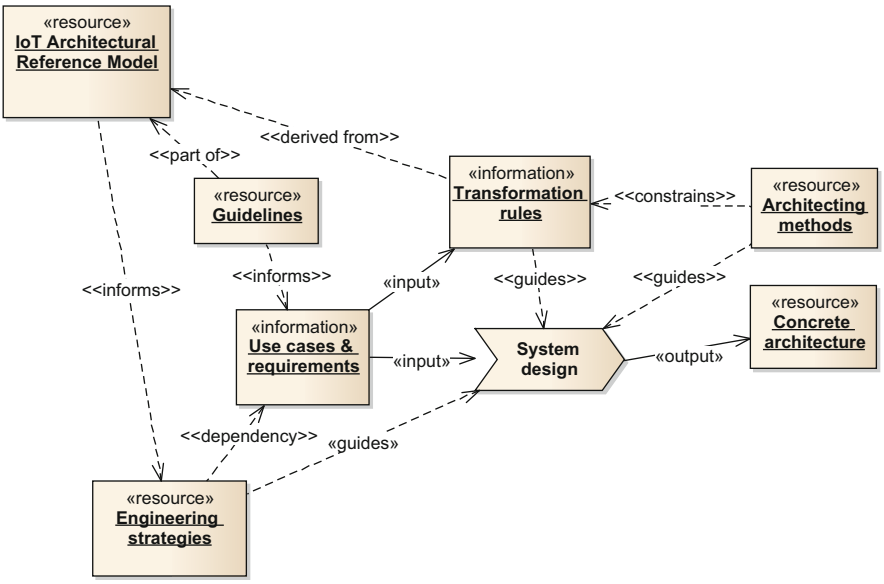

Fig. 3.3 Process for generating concrete architectures

Design and SysML. All of these methodologies were designed for very concrete use cases and application scenarios. Unfortunately, this high degree of specificity defines even their inner workings. In other words, if they are applied to generalised use cases, the result is not generalised models on the abstract level of an IoT ARM – in fact, the result is nothing. We illustrate this using the example of MDE.

MDE for the generation of Model-Driven Architectures (MDA) is standardised by the *Object Management Group* (OMG) (Miller and Mukerji 2003). The main application area of this methodology is the development of software systems. MDE prescribes four steps for a development process:

1. Specify a system independently from the platform;
2. Specify platforms;
3. Choose a particular platform for the system;
4. Transform the system specification into that of the particular platform.

The goals behind this approach are portability, interoperability and reusability through the architectural separation of concerns (Vicente-Chicote et al. 2007). Thus, on the face of it, this all sounds very similar to the goals of our IoT ARM development process.

Figure 3.4 summarises the main idea of MDA. A platform-independent model, i.e. an architecture, is to be transformed into a platform-specific model, i.e. an implementation. An example for the former is a GUI user interface described in

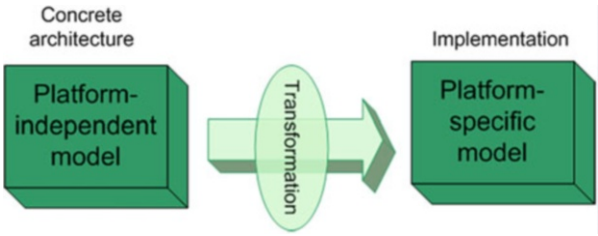

Fig. 3.4 Generalised architecture approach according to the Model-Driven Architecture methodology, otherwise known as Model-Driven Engineering (Miller and Mukerji 2003)

Table 3.1 Use of standardised architecture methodologies for the development of the IoT ARM

Methodology	Aspect adopted in our work
Aspect-oriented programming	Delineation of functionalities by aspects. This is embodied in the concept of functionality groups (see IoT Functional view in Chap. 8)
Model-driven engineering	General concept of transformation from a generic to a more specific model. We use this concept for describing and developing our guidelines
Views and perspectives	We adopt the concept of views and perspectives to derive the IoT Reference Architecture, i.e. we arrange all aspects of our reference architecture according to views and perspectives (see IoT Reference architecture in Chap. 8)

UML, and the latter is an implementation of this interface in a mobile phone model featuring a particular operating system.

This sounds very much like the transformation introduced in Fig. 3.3, but it actually takes place at a lower abstraction level, as becomes apparent from Fig. 3.2. The IoT ARM and the MDE approach are thus linked to each other through platform-independent models (architectures). While the general idea of a model transformation, as promoted by MDE, resonates with our IoT ARM approach, the methodology developed for deriving transformations between platform-independent and platform-specific models can, alas, not be transferred and adapted for deriving best practice transformations.

Table 3.1 summarises how we use ideas borrowed from standardised architecture methodologies for our work on the higher abstract level of our IoT ARM.