An IoT Solution Methodology

Frank Puhlmann, Dirk Slama

Bosch Software Innovations GmbH
D-10785 Berlin, Germany
{frank.puhlmann,dirk.slama}@bosch-si.com

Abstract. Internet of Things (IoT) projects extend traditional projects, since they combine four different domains: A Product Design & Manufacturing project for the hardware part, an Embedded IT project for the firmware, a Telecom project for the communication aspects, as well as a traditional Enterprise IT project for the backend and integration part. To link these domains together and derive a common understanding between all stakeholders, we propose an IoT solution analysis methodology that is based on business processes as key artefacts. This methodology has been derived from practice over the last three years and applied in recent practical IoT projects.

1 Motivation

The Internet of Things, abbr. IoT, has gained a lot of coverage in the last years [3, 4, 13]. It is now a common understanding that the implications of this development will have a strong impact on businesses and individuals [15, 5, 12]. Products will be enhanced with connected services and new business models implemented on top. Market research companies estimate the number of connected devices into two-digit billions by 2020 [9] and the possible revenue streams accordingly [6].

In practice, however, IoT projects are not so well understood as the hype suggests. IoT projects tend to be split across multiple domains [18], making them hard to execute successfully. First of all, IoT projects have a hardware part, which is usually a complete project on its own, either retrofitting existing or developing custom things. Since the hardware needs to be able to communicate and interact, usually a complex firmware is deployed to it, which needs to be developed and maintained with a different lifecycle. If the connectivity goes beyond Bluetooth or Wifi, mobile operators with SIM cards and yet other lifecycles come into play. Finally, connected things usually require mobile apps or backend services to operate, which need to be developed and operated in the sense of a traditional IT project.

Bringing these four different lifecycles—and often also projects—together requires a structured approach. Issues like waterfall-driven hardware developments, fixed SIM lifecycles, complex certification of firmwares, and of course the agile, often Dev-Ops driven backend IT project need to be aligned in the right way. We covered this topic in depth in our book *Enterprise IoT* [18][1].

[1] Available online at http://enterprise-iot.org/book/enterprise-iot/

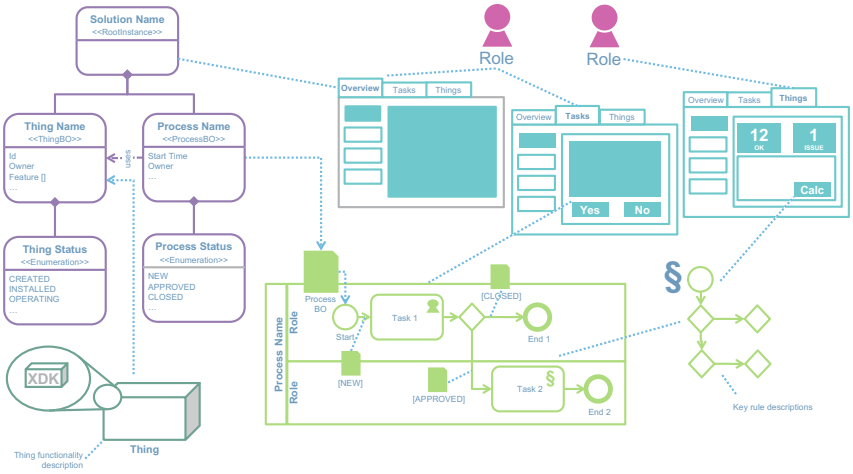


Fig. 1. The IoT Solution Draft Template.

In this paper, we go beyond the books solution delivery approach denoted as *Ignite* and propose a hands-on IoT methodology to analyse the initial requirements of all involved stakeholders. This IoT Solution methodology assumes that the use case or focus area has already been identified. It stops right before a concrete architecture decision, allowing for different implementation strategies and vendors. The methodology itself is build on proven concepts like use case analysis and UML class diagrams with a strong focus on business processes.

Figure 1 shows the structure of the basic IoT Solution Draft template. This template is used in initial IoT project workshops and created together with all required stakeholders. In contrast to existing approaches, it provides an integrated view of all artefacts that can be broken down into the different sub-projects later on. The draft focusses on the key stakeholders, the high-level domain model, the business processes and rules, user interfaces and patters, as well as—of course—things and connectivity. All aspects will be broken down in subsequent drafts.

The following paragraphs introduce the required preliminaries in section 2, formalise the concepts in a meta-model in section 3, and discuss them by example in section 4. An aggregating template denoted as *Solution Sketch* will be introduced in section 5. Section 6 discusses best practices and related work.

2 Preliminaries

The IoT Solution Methodology re-uses a number of existing standards and best practices due to their widespread tool availability and practical experiences. While most of the standards will be simplified for the sake of usability in practice workshops, the simplification is done by restricting the used elements, so that they are still fully compliant.

Before we move to the standards, however, there is one more term: In the introduction we talked about *things*, but we need to detail them for the next sections. We use the word *thing* to depict the concept of anything that can be connected, either physical or virtual. The refinement is given by the term *asset*:

Preliminary 1 (Assets and Devices). An *asset* (as defined in Enterprise IoT in [18]) is a thing of value for a company, i.e. something a company keeps in its books, like the cars a rental company owns. An assets can have multiple *devices* attached that collect data, act on something, or process and transmit the data.

Preliminary 2 (UML Class Diagrams). A core concept of the IoT methodology is given by a subset of UML Class Diagrams, see e.g. [16]. We denote this subset as *Domain Models*. A *Domain Model* always has a single root class with a specific stereotype *RootInstance*. This class is used as a start for navigating the model in queries and expressions. Other stereotypes are used as introduced in section 4. For simplicity, we do not display the methods as well as attribute types in the class body, focussing purely on the attributes. We restrict the associations to inheritance, aggregation, and directed references.

Preliminary 3 (Petri Nets). The lifecycles of *Domain Model* classes are described via Petri nets, as defined in e.g. [7]. The primary use is restricted to the standard visualisation with places and transitions to allow for token games with the stakeholders. Besides this, all formal rules and formalisations still apply and might be used for further analysis or implementation later on.

Preliminary 4 (Business Process Diagrams). The process modelling part of the methodology is based on the BPMN 2.0 specification as defined in [1]. We restrict the usage to the most common elements as described in [21] but allow for any addition if understood by the stakeholders. The common elements include plain *Start* and *End Events*, *Human* and *Service Tasks*, as well as *Exclusive* and *Parallel Gateways*. We use the *Data Object* to link with a specified class of the *Domain Model*.

3 The IoT Solution Draft Meta-Model

The elements of the *Solution Draft* are depicted in a meta-model shown in figure 2. The *Solution Draft* class groups the elements from figure 1, namely *Rules*, *User Interfaces*, *Roles*, *Processes*, the *Domain Model* including an *Analytics Model* not shown in figure 1, the *Things* as well as the corresponding *Connectivity* concept. Furthermore, the most important associations are shown to depict the connections between the different elements.

We describe the logic behind the elements and associations in the following paragraphs. A detailed discussion of the elements follows in section 4 by example. We omit obvious extensions to the meta-model like *User*s etc. for the sake of simplicity. Furthermore, the meta-model might be enhanced with additional elements like *Report*s etc.

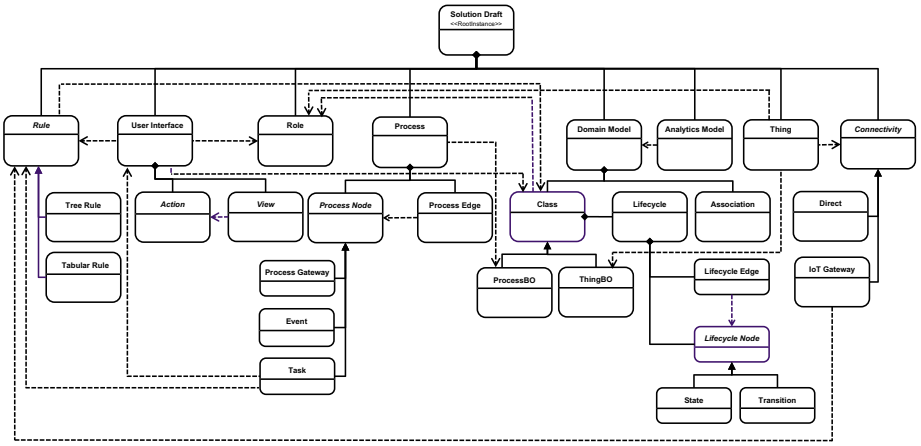


Fig. 2. The IoT Solution Draft Meta-Model.

Element 1 (Rule). *Rules* usually describe stateless logic required for an IoT solution. We distinguish two variants, either tree- or tabular-based. *Rules* need to reference *Classes* of the *Domain Model* to describe their input and output values.

Element 2 (User Interface). *User Interfaces* group *Actions* and *Views*. An *Action* is a trigger like a button or slider. A *View* contains a layout for attributes of the *Classes* referenced by the *User Interface*. A typical view might have a list, tabular, or textual layout. *User Interfaces* can also reference *Rules* and *Roles*. The former describes additional logic, like the flow of a wizard via a *Tree Rule*. The latter describes access rights, like which *Role* is allowed to view or edit elements shown in the *User Interface*.

Element 3 (Role). A *Role* represents a stakeholder of the IoT solution with its corresponding access rights. *Roles* are mapped to *Users*, which we omit for simplicity.

Element 4 (Process). *Processes* are in the centre of an IoT solution. They are made up of *Process Nodes* and *Process Edges*. Again, for the sake of simplicity, we refer to standards like the BPMN for a complete subset. We depict *Process Gateways*, *Events*, and *Tasks* in the meta-model. *Tasks* can reference *User Interfaces* for their visualisation (BPMN User Task) as well as *Rules* (BPMN Rule Task) to represent the execution of stateless logic inside a process flow.

Element 5 (Domain Model). The *Domain Model* describes the logical business view of the IoT solution data. A *Domain Model* aggregates *Classes* and *Associations*. *Classes* can be refined into specializations used as transitional data inside *Processes* or data representing a *Thing*. *Classes* can also aggregate *Lifecycles*, which describe the state transitions a class can have. In terms of UML, a *Lifecycle* is usually depicted as an enumeration of the different states.

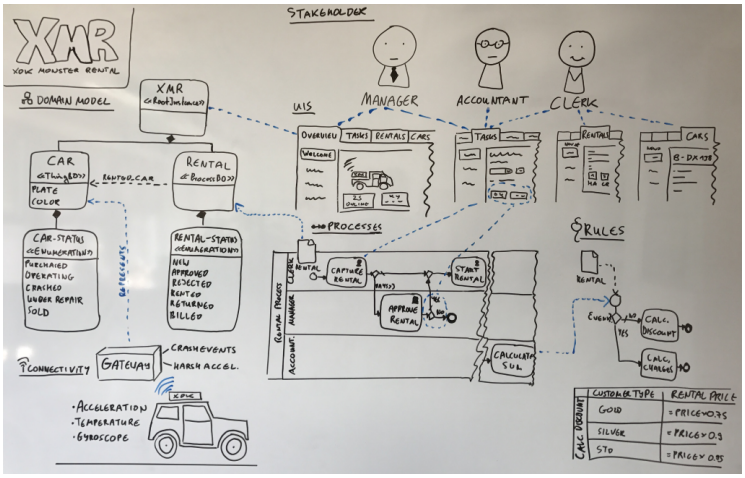


Fig. 3. The Whiteboard Solution Draft.

The behavioural model behind the lifecycle is given by a set of *Lifecycle Edges* and *Lifecycle Nodes*, which resemble a Petri net with *States* and *Transitions*.

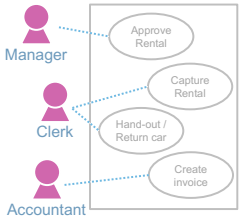
Element 6 (Analytics Model). An *Analytics Model* references elements of the *Domain Model* to describe goals and computations for machine or deep learning algorithms used in areas like predictions or self learning of IoT solutions.

Element 7 (Thing). A *Thing* represents the physical centrepiece of an IoT solution. It can reference *Roles*, which describe the rights to own or use (aspects) of the *Thing*. A *Thing* used in an IoT solution also needs to have a reference to at least one *Connectivity* model.

Element 8 (Connectivity). *Connectivity* describes how *Things* interact with the backend systems. Typical communications are either *Direct* or *IoT Gateway* based. The former describes direct connectivity with the backend, like a GSM-enabled car emergency sensor. The latter uses a local device which groups several *Things* and provides local logic via onboard *Rules*, such as given by a Smart Home gateway.

4 The IoT Solution Draft by Example

This section illustrates the introduced meta-model with the practical example of a fictional car rental company called *Monster Rentals*. The company plans to introduce a pricing based on the driving behaviour of the customer. In addition to the standard fee, the driver will be charged for harsh accelerations and get a discount if the ride goes smooth. Furthermore, a crash sensor will immediately inform the rental company to take actions in case an accident happens. Therefore, the rental cars will be equipped with onboard units who captures the driving behaviour and transmit the sensor data in real time to the company. The



Name	Description	Count Initially	Count Planned
Manager	Is responsible for the operations of the rental company office.	1	50
Clerk	Is responsible for customer care and rental support.	3	200
Accountant	Is responsible for creating the invoices.	1	10

Fig. 4. The Roles refinements of the Solution Draft.

backend systems will take care of the booking and invoicing processes. We omit a user self-service app for simplicity.

The initial creation of the Solution Draft took place on a Whiteboard and was captured in figure 3. The following subsections will revisit and break-down the different elements contained in the Solution Draft.

4.1 Stakeholders and Roles

The *Solution Draft* shows the most important *Roles*, usually on top. The underlying discussion starts earlier with the *Stakeholders*, which are split into four areas: The *Solution Users*, the *Asset Users*, the *Enterprise Users*, as well as the *Partner Ecosystem Users*. After having identified them, the relation between the areas are drawn, e.g. *Asset Users* interacting with *Enterprise Users*.

The refinement of the *Roles* is shown in figure 4. A good way to start is using graphical UML Use Case diagrams to capture the key use cases. An important additional step is capturing the number of users, both initially and for the planned rollout. This information is important to define the scale of the architecture later on.

4.2 Domain Model and Lifecycles

The *Domain Model* shows the conceptual view of the solution from a business view as well as aligning the different other models. For instance, any field shown in a mockup UI should have a corresponding attribute or class, or any thing's property should be represented as well. The same holds for thresholds and process variables etc. The running example for the rental company is given in figure 5. As stated in preliminary 2, we use specific stereotypes:

- *ProcessBO*: Represents a business object that is routed through a process. In our rental company, this is the rental process object containing details like the start time or the price.
- *ThingBO*: A business object representing a thing, e.g. a car in our example.

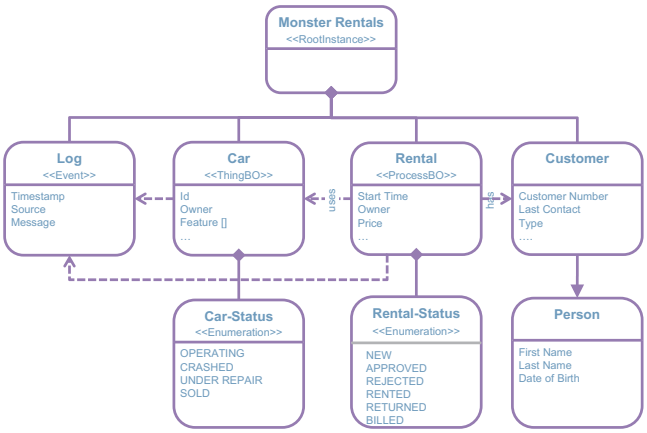


Fig. 5. The Domain Model refinement of the Solution Draft.

- *Event*: An event that might occur during the execution of a process or directly come from a thing. An event instance is usually used as some kind of log.
- *Enumeration*: An enumeration is primary used to capture the different states of a thing or process business object but can also be used for simple enumerations like the available colours or car types.

To refer to any entity in the *Domain Model*, we use a simple arrow/dot notation starting from any class directly aggregated by the *RootInstance*, denoted as *Path Expressions*. Formally:

```
PathExp   ::= ClassName->PathExp | ClassName |
              ClassName.Attribute
ClassName ::= <Name of a Class>
Attribute ::= <Name of an Attribute>
A->B      :   Denotes a direct association between
              class A and B
A.attr    :   Denotes a direct attribute of class A
```

Examples:

- `Rental.StartTime` refers to the start time of any rental; can be filtered like `Rental.StartTime=Today` (not formalised here)
- `Rental->Customer` refers to any Customer

There is a special case, where you don't need to filter your entities: A *ProcessBO* class used in a *Business Process* or *Rule* always refers to the instance

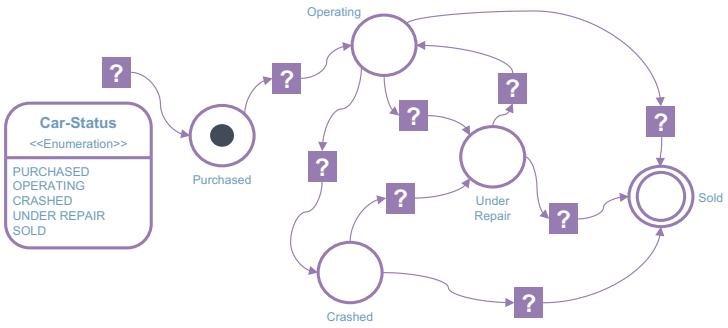


Fig. 6. The Lifecycle refinement of the Solution Draft.

it is bound to. We will discuss this later when we talk about business processes and rules.

Figure 6 shows the *Lifecycle* (e.g. behavioural) view of the solution based on classes of the *Domain Model*. Especially, each *ThingBO* and *ProcessBO* need to have an aggregated *Enumeration* representing the different states the business object can have. The figure shows the status of the *Car*, which starts with *Purchased* from the rental's company point of view. It has some intermediate states before reaching the final state *Sold*. As stated in preliminary 3, a *Lifecycle* is a Petri net with a special visualisation of the final place. Each state given by the attributes of the *Enumeration* are mapped to a place of the Petri net. The transitions need to be added accordingly as discussed with the *Stakeholders*. A technical implementation will be derived later on, as discussed in the next subsection.

4.3 Processes and Rules

After having defined the lifecycles of the *Business Objects*, we need to model the *Business Processes* implementing them. In general, two different ways of implementing the *ProcessBO Lifecycles* via *Processes* are possible: (1) Processes are the only mean to achieve state transitions of the business objects or (2) processes are the recommended or optional way to achieve the transitions. The former is usually understood as a process-driven application [19]. The latter is known as case-driven [10].

As stated in preliminary 4, the modelling of the *Processes* is done according to the *BPMN 2.0* standard. We usually restrict the use to single *Pools* with *Lanes* for the different *Roles*. Typically, three different *Task* types are sufficient: (1) *Human Tasks*—a task performed by a human via a frontend; (2) *Service Tasks*—a task performed by a system; and (3) *Rule Tasks*—a calculation executed by a *Rule* engine. Furthermore, the *Process Gateways* can be restricted in many models to *exclusive* and *parallel* behaviour. The *Business Objects* of the *Lifecycle* are linked as discussed in the next paragraph by example.

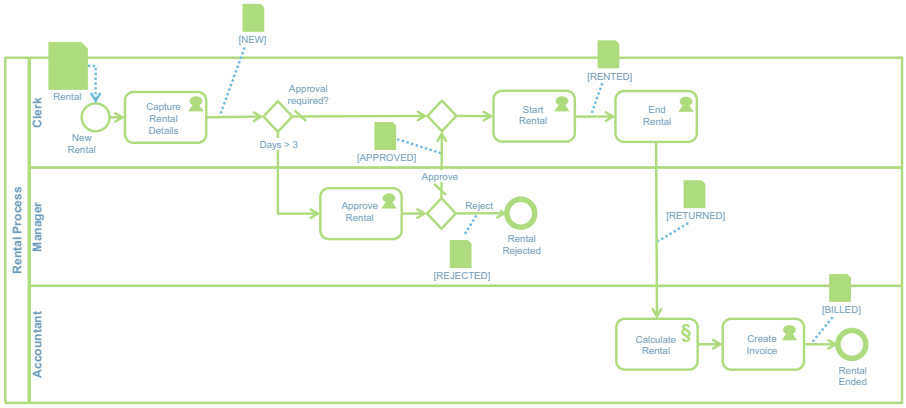


Fig. 7. The Process refinement of the Solution Draft.

Figure 7 shows a Business *Process* of the Car Rental company. The process captures the basic *Rental Process* as executed by a *Clerk* of the company. To visualise the *ProcessBO* used, it is associated graphically with the *Start Event*. If the completion of an *Activity* changes the *Lifecycle* state of the *Business Object*, the outgoing *Sequence Flow* of the activity associates the *ProcessBO* with the corresponding *State* shown in brackets.

The verify the modelled *Business Process*, we step through the *Process* and check that only *Lifecycle Transitions* are possible that are captured in the *Lifecycle Model*. If the *Business Process* contains more behaviour, we need to discuss with our *Stakeholders* if the *Lifecycle Model* or the *Business Process Diagram* need to be adjusted. If the *Lifecycle Model* allows more behaviour than is captured in a single *Business Process*, this points to a case-driven implementation. The additional transitions might be captured in other *Business Processes* or handled via process-external logic and user interface implementations.

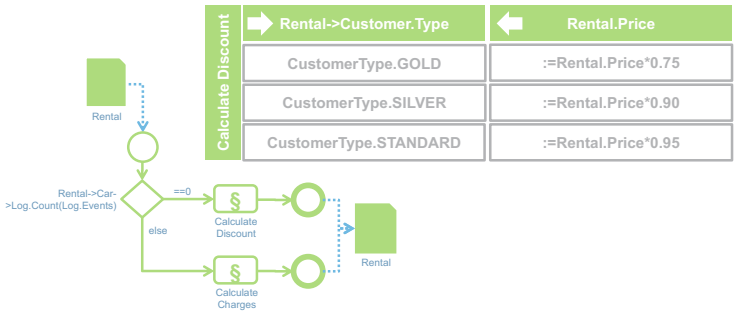


Fig. 8. The Rules refinement of the Solution Draft.

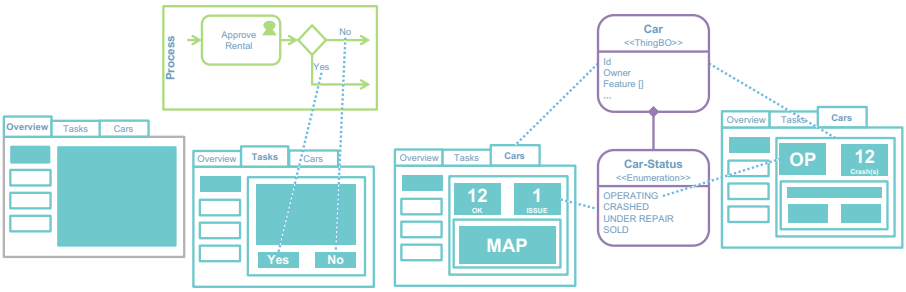


Fig. 9. The User Interface refinement of the Solution Draft.

While the *Business Processes* as introduced are usually long-running, *Business Rules* typically describe stateless functional business logic. Like *Business Processes*, *Business Rules* work on *Business Objects*. More recently, the OMG standardised a notation for *Business Rules*, the *Decision Model & Notation* (short DMN, see [2]). Since the DMN has not gained the traction that BPMN has in the acceptance and tools support, we typically recommend a simpler subset: (1) Tree-based rules; e.g. if-the-else style and (2) Tabular rules; e.g. Excel-style. Both styles are shown by example in figure 8.

A tree-based *Rule* takes an entity from the *Domain Model* as an input. Decisions are denoted as diamonds and possible results are gated by *Path Expressions* as introduced in subsection 4.2. A tree-based *Rule* is also used to orchestrate the tabular rules, like the calculation of the discount in the example. The table representation defines how input parameters (like `Rental->Customer.Type`) are mapped to an output parameter (e.g. `Rental.Price`) via *Path Expressions*.

In general, *Business Rules* make *Business Processes* more flexible, since they have an independent deployment lifecycle, which is usually more easy to change and adapt. They also help in centralising and re-using business logic in different *Business Processes* or other parts of the *Solution*.

4.4 User Interfaces and Patterns

This subsection focuses on a short recap of the core techniques to map *User Interfaces* with the models and artefacts shown before. A refined discussion on how to map *Process and Rules* with *User Interfaces* can be found for instance in [11]. Still, it has proven valuable to stick to three core principles:

- *Use Black/White schematics*: A good *User Interface* should visualise everything if mapped to only two colours.
- *Keep it simple*: Think twice if you really need each element.
- Focus on functionality: Each element should have a clearly defined function that is the same where ever it appears.

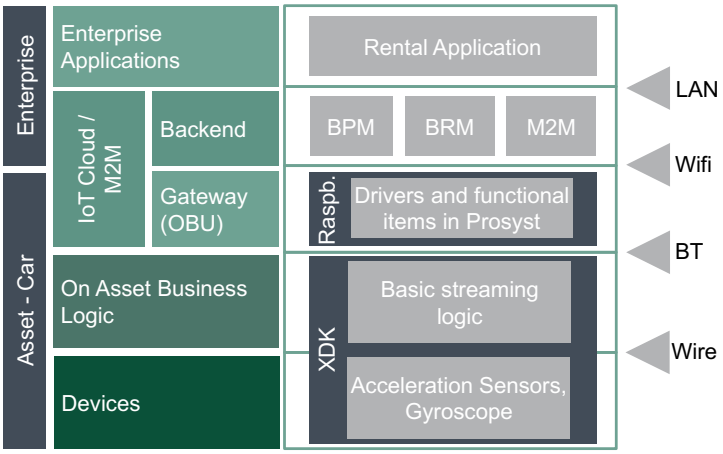


Fig. 10. The Connectivity refinement of the Solution Draft.

Figure 9 shows how *User Interfaces* are linked with *Processes* and entities from the *Domain Model*. For instance, each *Human Task*, where a decision is made, must be reflected by corresponding *Actions*, e.g. buttons, in the UI. The *UI* buttons must be labeled the same as the outgoing *Sequence Flows* of the *Process Gateway* following a *Human Task*. *User Interfaces* representing entities from the *Domain Model* are required to clearly distinct between collections (like a list of cars) and single entities (like the details of a car). Typically, the bottom of the *UI* should show more details and group it by functionality or features.

4.5 Assets and Connectivity

A core concept of the IoT methodology is the *Asset Integration Architecture*, short AIA, taken from our *Ignite methodology* [18]. In contrast to other elements of the *Solution Draft*, the AIA already considers possible implementation details provided by the Telco- and Communications project stream. The example AIA is shown in figure 10.

Basically, an AIA is split into two sections, the Asset and the Enterprise part. Which one is top, usually depends on the customer: If the stakeholders are Enterprise-centric the Enterprise section should be on top, while Asset-centric stakeholder would model the Asset on top.

Investigating the sample AIA from top to bottom, the core sub-elements are the Enterprise Applications and the IoT (Cloud) or M2M implementations, which might be split into a backend or gateway part. In the IoT world, an *IoT Gateway* is typically used to connect assets, which (1) have no direct Internet connectivity build in; e.g. simple door sensors etc or (2) require local business logic if no Internet connectivity is available; e.g. a car passing through a tunnel. The asset itself is composed of devices (sensors, actuators) and some on asset business logic like the detection of an event that needs to be sent.

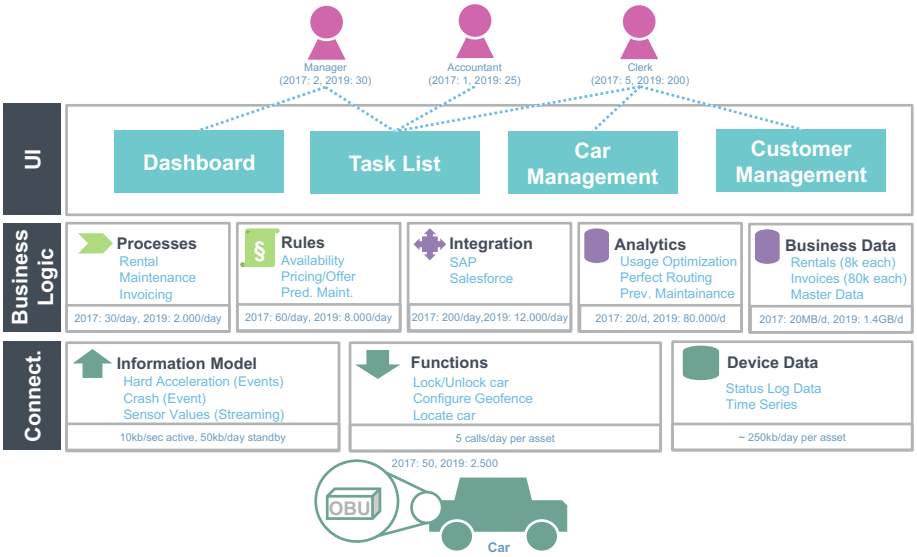


Fig. 11. The final Project Sketch.

In our Rental scenario, we use a *Business Process Management System*, a *Business Rules Management* component, as well as an *M2M* system. Since these systems and components run on backend servers, they are all connected by a *Local Area Network* (LAN). To implement the *IoT Gateway*, we use a Raspberry Pi for the demo scenario, which runs a commercial M2M stack that provides drivers and business logic (functional items) for the scenario. The Raspberry Pi is connected via *Wifi* with the backend and via *Bluetooth* with the on asset business logic. For a running prototype, we use a Bosch XDK (see https://xdk.bosch-connectivity.com) that provides sensors for acceleration as well as a gyroscope that we use to detect crashes.

5 The IoT Project Sketch

The final activity discussed in the scope of this paper is the creation of a *Project Sketch* from the detailed refinements of the *Solution Draft*. The *Project Sketch* is shown in figure 5 and used to give a one-page overview of a proposed project to a project sponsor.

In contrast to a *Solution Draft*, a *Project Sketch* describes the implementation aspects from three logical layers: *User Interfaces*, *Business Logic*, as well as *Connectivity*. Each layer's components can be directly translated to a corresponding technology. The *User Interface* layer can be implemented for instance in Desktop *Java* or with *Mobile Apps*. The *Processes* should be implemented with a *Business Process Management System*, which might also include the *Integration* aspects. The *Rules* should be implemented with a *Business Rules Management System*.

Different approaches can be used for the *Analytics* requirements, incl. *Complex Event Processing* or *Machine Learning Technologies*. The *Business Data* is usually stored in *SQL* databases. The *Communication* layer is implemented by a an *M2M* stack. Furthermore, since the *Project Sketch* is meant as an overview of a concrete next project, only the topmost *Processes*, *Rules* etc. are listed.

Just like the *Solution Draft*, the *Project Sketch* starts with the most important *Roles* and their mapping the corresponding *User Interfaces* of the *Application*. In addition to the *Solution Draft*, each artefact contains the scales by numbers, e.g. *Users* of a specific *Role* this year and next year. The same holds for all other aspects, like the number of *Process* and *Rule* instances, or the size of the *Business Data*. These numbers give an indication on the scale of the final solution.

While the derivation of the top and middle layer from the *Solution Draft* is straightforward, the *Connectivity* layer needs some additional explanations. First, the *Information Model* captures the high level events and data streams coming from the things; usually implemented in the *On Asset Business Logic* or the *Gateway/Agent* of the *Asset Integration Architecture*. Second, the *Functions* cover functionality that needs to be implemented on the things. Third, the *Device Data* is usually stored in *No-SQL* databases to store time series and log date coming from the things. All described refinements capture the expected input/output or data size, scaling via the asset numbers shown in the sample *Thing* below.

6 Evaluation and Related Work

We use the *Solution Draft* and *Project Sketch* in our day-to-day operations to analyse most customer inquiries. Usually, we receive inquiries via a *Request for Proposal*, short RFP. We use the information provided in the RFP to derive a *Solution Draft* based on team-internal discussions. Almost anytime, the RFP lacks information that is required for a complete analysis. The second step is hence an on-site customer workshop, where we discuss the analysis done with the customer and try to gather the missing information. In some instances, we start immediately with a customer workshop, usually if we are already involved in the IoT ideation and business model creation. The next subsection introduces best practices collected from multiple projects as well as discusses related work.

6.1 Evaluation

Based on our experience, we derived the following best practices from projects with customers from multiple domains, including manufacturing, retail, insurance, and mobility.

Best Practice 1 (Use the Solution Draft as a Canvas). The *Solution Draft* showed value in structuring the available whiteboard space into different regions. Regardless with which aspect the customer starts, the draft helps drawing it in the right place with space left for the other aspects.

Best Practice 2 (Use the Solution Draft as a Guide on the Required Inputs). In our daily practice, we often observe that customers are not aware of all the different aspects required for an IoT solution. The *Solution Draft* helps them to see the blind spots directly on the canvas. This often leads to an early and open discussion on which requirements need to be discussed in detail from the customer's end.

Best Practice 3 (Detail as needed). Use the refinements of the *Solution Draft* as required. Usually, the refinements are done within specific workshops and selected team members only. A detailed discussion on the different domains (hardware-, embedded-, telco, enterprise-project) can be found in [18].

Best Practice 4 (Application areas). In our experience, the *Solution Draft* methodology is used best for individual projects implemented with standard tool suites. We successfully applied it for mobility, retail, insurance, and industrial monitoring solutions. If the customer expects an out-of-the box solution, e.g. often found in areas of Industry 4.0, the *Solution Draft* does work, but a more focussed configuration of the packaged solution often fits better.

Best Practice 5 (Project Sketch). Use the *Project Sketch* to propose a concrete project to a sponsoring stakeholder. The *Project Sketch* makes it easy to derive vendor-specific implementations of IoT solutions. Furthermore, the scale of the implementation is visualised and can be directly discussed with the stakeholders. Many product managers, for instance, do not foresee the vast amount of data that is produced by the things that needs to be stored, either by specification or legal requirement.

6.2 Related Work

Structured methodologies for analysing problems in general or specific domains exists almost as long as computer science. For instance, an early approach to formally capture behaviour is given by Petri nets [7]. The Unified Modelling Language (UML) is mostly known for its Class Diagrams, which capture the static structure of a software system [16]. Domain-specific notations include for instance the EPK standard used by the ARIS business process methodology [8]. Newer standards include the *Business Process Model and Notation* (BPMN) standard [1].

To the best of our knowledge, there exists no vendor-independent standard for modelling and analysing IoT solution yet. We wrote an early introduction to an IoT Project methodology denoted as *Ignite* [18]. This methodology, however, has a strong focus on the project execution and organisation and only partly focused on the analysis. Some concepts, like the *Asset Integration Architecture* have been taken from this work.

Other authors, however, discuss an IoT methodology from different angles or use cases besides modelling and analysis as we did. For instance, V. Sharma et.al. discuss a design-thinking based approach in [17]. Their paper focusses on the discovery of use cases for IoT scenarios. Perera et.al. introduce a privacy-by-design framework for IoT applications in [14], which focuses on adapting best

practices for privacy design into the IoT domain. A framework for provisioning large scale, scalable IoT deployments, such as in Smart Cities, is introduced by Voegler et.al. in [20]. This work, however, focuses on a concrete architecture, whereas our methodology is architecture agnostic.

7 Conclusions

This paper introduced a novel IoT methodology based on a meta-model that brings together the different aspects typically found in IoT projects. The IoT methodology is based on a visual template for the high level structure, denoted as *Solution Draft*, as well as for detailed discussions. The templates follow existing notations wherever possible. The outcome of the analysis is captured in a concrete decision proposal (*Project Sketch*) for a project.

From our evaluations and practical experience, this methodology unfolds its value especially in a *Plan-Build-Run* environment. After having done the *Planing* via the different refinements discussed in section 4, multiple *Build* options can be derived. A common implementation might be cloud-based with a *Node.js* template, while some on-premise implementations might generate traditional *Java2EE* code. In general, the IoT methodology is architecture agnostic.

Future work focuses on enhancing the methodology regarding and on building different implementation-support tools. These tools will be based on the *meta-model* introduced in section 3. Focusing especially on the *Processes* and the *Domain Model*, rapid-prototypes can be build. Using agile implementation practices, these prototypes will enable early user feedback and support UX processes.

Acknowledgements. The authors would like to thank Amro Al-Akkad for his valuable feedback on the methodology.

References

1. Business Process Model and Notation (BPMN) version 2.0. OMG Specification, Object Management Group (2011)
2. Decision Model and Notation (DMN) version 1.2. OMG Specification, Object Management Group (2016)
3. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A Survey. Computer networks 54(15), 2787–2805 (2010)
4. Da Xu, L., He, W., Li, S.: Internet of Things in Industries: A Survey. IEEE Transactions on industrial informatics 10(4), 2233–2243 (2014)
5. Fleisch, E., Weinberger, M., Wortmann, F.: Business Models and The Internet of Things. In: Interoperability and Open-Source Solutions for the Internet of Things, pp. 6–10. Springer (2015)
6. Ip, C.: The IoT opportunity: Are you ready to capture a once-in-a lifetime value pool? Hong Kong IoT Conference (2016)
7. Jensen, K.: Coloured Petri Nets. Springer Verlag, Berlin, 2nd edn. (1997)
8. Keller, G., Nüttgens, M., Scheer, A.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Tech. Rep. 89, Institut für Wirtschaftsinformatik, Saarbrücken (1992)

9. Lucero, S.: IoT platforms: enabling the Internet of Things. IHS Technology Whitepaper (2016)
10. Meyer, A., Herzberg, N., Puhlmann, F., Weske, M.: Implementation framework for production case management: Modeling and execution. In: 18th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2014, Ulm, Germany, September 1-5, 2014. pp. 190–199 (2014)
11. Nelius, R., Slama, D.: Enterprise BPM: Erfolgsrezepte für unternehmensweites Prozessmanagement. dpunkt. verlag (2011)
12. Pelino, M., Gillet, F.: The Internet of Things Heat Map, 2016. Forrester Research (2016)
13. Perera, C., Liu, C.H., Jayawardena, S., Chen, M.: A survey on internet of things from industrial market perspective. IEEE Access 2, 1660–1679 (2014)
14. Perera, C., McCormick, C., Bandara, A.K., Price, B.A., Nuseibeh, B.: Privacy-by-design framework for assessing internet of things applications and platforms. In: Proceedings of the 6th International Conference on the Internet of Things. pp. 83–92. ACM (2016)
15. Rifkin, J.: The zero marginal cost society: The Internet of Things, the collaborative commons, and the eclipse of capitalism. Palgrave Macmillan (2014)
16. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, Boston, 2nd edn. (2005)
17. Sharma, V., Das, S., Kewaley, S.: Design thing'ing: methodology for understanding and discovering use cases in iot scenarios. In: Proceedings of the 7th International Conference on HCI, IndiaHCI 2015. pp. 113–115. ACM (2015)
18. Slama, D., Puhlmann, F., Morrish, J., Bhatnagar, R.M.: Enterprise IoT: Strategies and Best Practices for Connected Products and Services. O'Reilly Media, Inc. (2015)
19. Van Der Aalst, W.M., Ter Hofstede, A.H., Weske, M.: Business process management: A survey. In: International conference on business process management. pp. 1–12. Springer (2003)
20. Vögler, M., Schleicher, J.M., Inzinger, C., Dustdar, S.: A scalable framework for provisioning large-scale iot deployments. ACM Transactions on Internet Technology (TOIT) 16(2), 11 (2016)
21. Wohed, P., Aalst, W., Dumas, M., Hofstede, A., Russell, N.: On the Suitability of BPMN for Business Process Modelling. In: Dustdar, S., Fiadeiro, J., Sheth, A. (eds.) Business Process Management, volume 4102 of LNCS. pp. 161–176. Springer Verlag, Berlin (2006)