future internet

Article

Agile Service Engineering in the Industrial Internet of Things

Thomas Usländer * and Thomas Batz

Fraunhofer IOSB, 76131 Karlsruhe, Germany; thomas.batz@iosb.fraunhofer.de
* Correspondence: thomas.uslaender@iosb.fraunhofer.de; Tel.: +49-721-6091-480

Received: 29 August 2018; Accepted: 4 October 2018; Published: 9 October 2018

Abstract: The emerging Industrial Internet of Things (IIoT) will not only leverage new and potentially disruptive business models but will also change the way software applications will be analyzed and designed. Agility is a need in a systematic service engineering as well as a co-design of requirements and architectural artefacts. Functional and non-functional requirements of IT users (in smart manufacturing mostly from the disciplines of mechanical engineering and electrical engineering) need to be mapped to the capabilities and interaction patterns of emerging IIoT service platforms, not to forget the corresponding information models. The capabilities of such platforms are usually described, structured, and formalized by software architects and software engineers. However, their technical descriptions are far away from the thinking and the thematic terms of end-users. This complicates the transition from requirements analysis to system design, and hence the re-use of existing and the design of future platform capabilities. Current software engineering methodologies do not systematically cover these interlinked and two-sided aspects. The article describes in a comprehensive manner how to close this gap with the help of a service-oriented analysis and design methodology entitled SERVUS (also mentioned in ISO 19119 Annex D) and a corresponding Web-based Platform Engineering Information System (PEIS).

Keywords: Service-oriented analysis and design; Industrial Internet of Things; Service Engineering

1. Introduction

The Industrial Internet of Things (IIoT) enables innovative smart applications by interconnecting assets within and between factories using Internet technologies [1]. Furthermore, the IIoT is connoted with the risk of disruptive business models as well as privacy and security concerns. The IIoT endangers the classical value chains as the interconnection and data acquisition of huge amounts of "things" enables new ways of (big) data analytics and solution spaces. However, there is less discussion regarding the ways in which the IIoT will also dramatically change the way software will be developed in the future in the light of emerging powerful IIoT software and service platforms. The analysis and design of industrial software applications cannot be isolated any more from the existing and future capabilities of these IIoT platforms. This concerns the requirements of the users who request remote accessibility of data and services via the Internet to support new business models (as well as the architecture) which need to be compliant to the current and future standards and products of the IIoT. Analysis and design methodologies for software applications shall take IIoT platforms into account.

The Industrial Internet Reference Architecture (IIRA) [2] describes the Industrial Internet as an "Internet of things, machines, computers and people, enabling intelligent industrial operations using advanced data analytics for transformational business outcomes". Industrial Internet systems cover multiple application domains, e.g., energy, healthcare, manufacturing, public sector, transportation, and related industrial systems. The IIRA asserts that they "must be easily understandable and supported by widely applicable, standard-based, open and horizontal architecture frameworks and reference architectures that can be implemented with interoperable and interchangeable building blocks". The German initiative Industrie 4.0 basically complies with these objectives but focuses on industrial production. As a starting point, a Reference Architecture Model Industrie 4.0 (RAMI4.0) was published [3].

Today, software and service engineering is performed in an agile manner in order to increase flexibility during the software design and development and to have milestones to re-focus the process. However, agility needs to encompass the requirements analysis phase too. The reason is that only in rare cases are the requirements fully available and fixed when software design and developments starts. To the contrary, especially in the domain of factory automation systems, requirements analysis is an agile process including a multi-step dialogue between the user(s), the stakeholders, and the software architects who know about the technological capabilities and constraints, and the project and development managers who may also estimate the effort to realize the expectations of the user [4]. Such an iterative analysis and design process leads to multiple reconsiderations and/or refinements of the user requirements as all parties involved learn from each other. If multiple users of one or even multiple organizations are involved, such dialogues are usually led during requirements analysis workshops facilitated by experienced system analysts or architects, and carried out occasionally by means of Web-based collaborative video conferencing systems.

This article describes a novel comprehensive methodology called SERVUS that, on the one hand, underpins this process, and on the other hand, accompanies and supports this process by a flexible documentation tool for all analysis and design artefacts in form of a Platform Engineering Information System (PEIS). SERVUS and PEIS explicitly take into account capabilities of IIoT service platforms as well as constraints of underlying production and manufacturing IT architectures, not to forget concerns about privacy and security.

1.1. Related Work

Software engineering methodologies that analyze and document user requirements in the form of use cases are state-of-the-art. Use case models describe the behavior of a system whereby, according to Jacobson and Ng [5], "a use case is a sequence of actions performed by the system to yield an observable result that is typically of value for one or more actors or other stakeholders of the system". Usually, use cases are modelled in graphical modeling languages such as the ISO/IEC 19505 Unified Modeling Language (UML) standard. Although quite illustrative, these UML models are often not precise enough if not associated with additional textual descriptions attached to the graphical symbols. However, if no further structure for the textual addition is prescribed, an interpretation of the use case idea is cumbersome. The idea to use semi-structured descriptions of use cases was first published by Cockburn in his book "Writing Effective Use Cases" [6].

With the trend towards service-oriented architectures (SOA) in the last two decades, various software engineering methodologies were developed that take SOA design principles into account. They are classified as service-oriented analysis and design (SOAD) methodologies which are basically service-oriented variants of model-driven architectures [4]. An overview and critical assessment is provided in Reference [7]. Typically, SOAD methodologies support the break-down from use cases to the specification of service requirements according to the constraints of service ecosystems. More recent comparison studies of software engineering methodologies were carried out, such as in Reference [8]. However, on the one hand, they all stayed on the conceptual level, and on the other hand, they all just focused on the top-down approach and ignored or at least neglected the fact that there may be already existing capabilities which may match the requirements.

Today, IIoT platforms exploit the potential that was prepared by the SOA approach. In addition to the provision of powerful and scalable storage systems and processing frameworks to handle big data challenges [1], IIoT platforms also offer pre-defined generic and domain-specific support services. For requirements analysis activities, this means that service requirements have to be mapped to service capabilities of IIoT platforms. Finally, this trend led to the fundamental question of how service descriptions may be matched on a semantic level. Several approaches in the research domain of semantic web services tried to solve this problem (e.g., Reference [9]), however, they could only provide partial solutions and none of them achieved practical relevance due to the inherent complexity of service matchmaking on a semantic level.

In the last years, comprehensive modeling frameworks such as Arrowhead [10] were specified and implemented that stressed the importance of rigid and comprehensive formal modeling of use cases, business activities, and system configurations, e.g., using UML, BPMN (Business Process Model and Notation) or SysML (Systems Modeling Language). The framework also considers emerging system-of-systems environments and finally IIoT environments [11]. In contrast to the service engineering methodology presented in this article, the Arrowhead frameworks focuses on the support of service discovery, authorization, and orchestration in operational SOA-based systems with the pre-requisite that the service requirements and capabilities are provided according to the modeling templates of the Arrowhead framework.

1.2. Structure of the Article

For the quick reader, the research results are summarized in Section 2. Section 3 describes the SERVUS methodology in more detail. Section 4 introduces the information system that supports the IIoT platform engineering, i.e., the Platform Engineering Information System (PEIS). Section 5 provides a discussion of the SERVUS methodology and the PEIS with respect to the best practices of Agile Modelling. The article concludes in Section 6 with a summary of the main concepts and a discussion about the next steps and the limitations of the methodology.

2. Results

The article highlights the importance of systematic agile service engineering for the Industrial Internet of Things and Industrie 4.0 software applications. Up to now, there is no software engineering methodology that covers systematically and concurrently both sides of a software development process necessary in an IIoT context: (1) a user-driven approach analyzing use cases and requirements, and (2) a platform-driven approach analyzing existing and missing capabilities and technologies of IIoT platforms. Complementary to Reference [12], which describes a former status of this research and only highlights a portion of the methodology, this article summarizes the whole methodology including a tool support.

For such applications, the functional and non-functional requirements of IT users (in Industrie 4.0 mostly from the disciplines of mechanical engineering and electrical engineering) need to be mapped to the capabilities and architecture patterns of emerging IIoT service platforms [13]. The capabilities of service platforms are usually described, structured, and formalized by software architects. An example is the three-tier architecture pattern of the IIRA and its functions structured into five functional domains (Control, Operations, Information, Application, and Business). Another example of such a platform comprises the services of the series of IEC 62541 OPC UA standards [14]. However, very often, such technical descriptions are far away from the expectations of the end-users and do not fit to their language and thematic terms. This complicates the transition from requirements analysis to system design, and hence the re-use of existing platform capabilities.

The article describes how to close this gap with the help of a novel service-oriented development methodology entitled SERVUS and a corresponding Web-based collaborative tool that supports its documentation. SERVUS denotes a Design Methodology for Information Systems based upon Service-oriented Architectures and the Modelling of Use Cases and Capabilities as Resources [4]. SERVUS considers use case models as being core artefacts of requirement analysis.

When designing industrial Internet applications, a use case expresses the functional, informational, and qualitative (i.e., non-functional) requirements of a user (i.e., an actor or a stakeholder) with respect to the system. Usually, use cases do not describe the user interactions themselves. It is essential for the use case description that the level of abstraction, the type of formalism as well as the language should be such that it is adequate to the domain of expertise of the users. In order to serve as a kind of contract with the user, a use case shall be both understandable to the user but also precise enough. Very often this means that use cases shall be specified in a non-technical way, normally achieved using plain text in natural language. However, in order to reduce the ambiguities and impreciseness of descriptions in natural language, structured textual descriptions are preferred. Use case descriptions are then structured according to a given template, e.g., an application form comprising identifier and description fields or thematic domain references associated with code lists.

The SERVUS design methodology recommends such a semi-formal description of use cases, basically following templates defined in Reference [6]. As an extension, this idea of a semi-structured description of analysis and design artefacts is also used when mapping the use cases to other design artefacts such as requirements and capabilities.

The SERVUS methodology was successfully used in numerous cooperative and inter-disciplinary software projects related to various application domains of the IIoT such as environmental information systems [4], environmental risk management, and early warning systems [15] as well as industrial production/Industrie 4.0 [12]. As a result of the domain-independent character of the IIoT and its emerging IIoT platforms, it may also be applied to interrelated application domains such as smart cities or smart regions. The SERVUS methodology is referred to by ISO 19119 as an example of a use case based-methodology for Geographic information Services [16].

3. SERVUS Methodology

The architectural style of service-oriented computing has already been present in practice in industrial software engineering for several years. However, although numerous methodologies for service-oriented analysis and design were described in the literature [4], there is not yet a standard methodology usable for the analysis and design of software applications based upon service-oriented IIoT platforms. The deficiency today is that there is no well-established methodology that explicitly supports a co-design of requirements and architectural design artefacts [17], i.e., a methodology that:

1. provides a two-sided integrated and parallel view of the requirements and the expert knowledge of thematic users with the services and information offerings of emerging IIoT platforms, and, in addition,
2. obeys explicitly the guidelines and constraints of architectural frameworks such as RAMI4.0 or IIRA as side-conditions.

Van den Heuvel et al. [18] call these aspects the open-world assumption that must be met by service networks. The "open-world" is characterized by "unforeseen clients, execution contexts and usage" of services operating in "highly complex, distributed, unpredictable, and heterogeneous execution environments". The SERVUS methodology tries to overcome these deficiencies. Its artefacts are organized according to the following three dimensions (Figure 1).

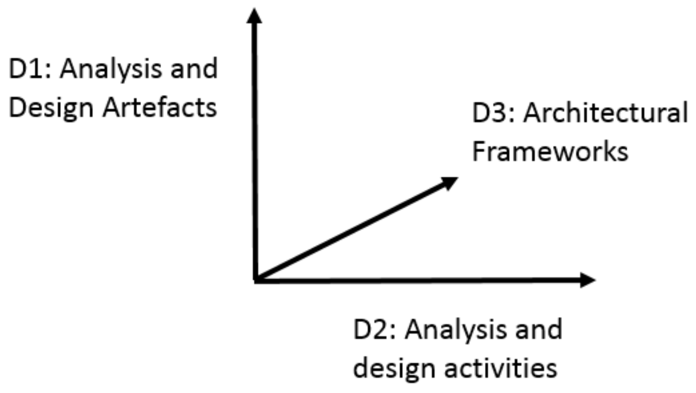
D1: Analysis and Design Artefacts
D3: Architectural Frameworks
D2: Analysis and design activities

Figure 1. Dimensions of the SERVUS methodology.

- D1: Analysis and design artefacts (i.e., the SERVUS meta-model, see Section 3.1)
- D2: Analysis and design activities (see Section 3.2)
- D3: Architectural frameworks, i.e., relationship to reference architecture models (such as RAMI4.0 and IIRA, see Section 3.3)

3.1. D1: Analysis and Design Artefacts

3.1.1. SERVUS Meta-Model

The SERVUS meta-model comprises the following analysis and design artefacts (see Figure 2):

- User story (US): description consisting of one or more sentences in the everyday or business language of an actor that captures what a user does or needs to do as part of his or her job function (cf. Wikipedia). A user story captures the 'who', 'what', and 'why' of a requirement in a simple, concise way.
- Use case (UC): description of core artefact of requirement analysis. Use case models describe the behavior of a system whereby "a use case is a sequence of actions performed by the system to yield an observable result that is typically of value for one or more actors or other stakeholders of the system". They describe the functional aspects, actors (user roles), and workflows.
- Requirement (REQ): description of functional and non-functional requirements of the system under design that may support the execution of a use case.
- Capability (CAP): description of existing and future capabilities of the system under design that realizes requirements. Capabilities are described independently of the technologies that may be used to implement the capability.
- Technology (TECH): description of the existing or emerging technologies and products that are available or expected to appear on the market. Technologies and products are distinguished by an attribute.

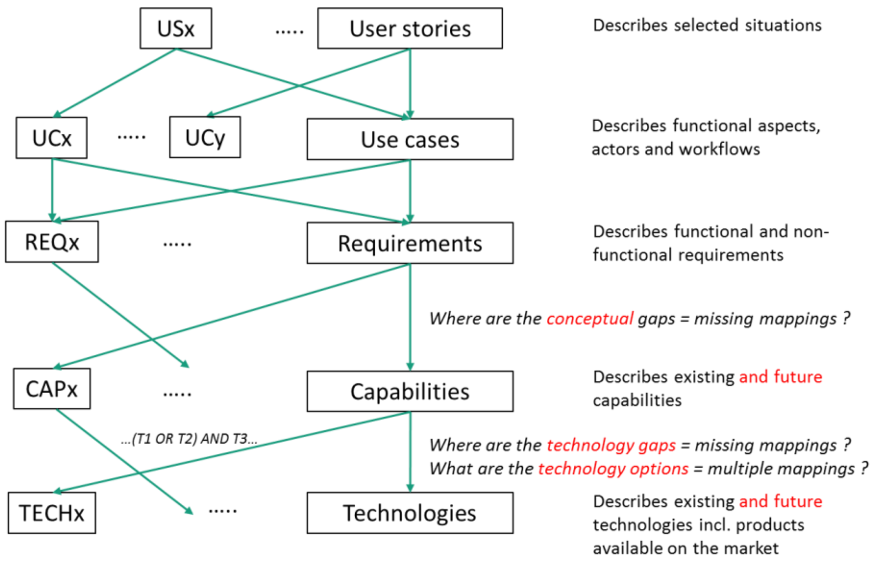
USx
.....
User stories
Describes selected situations
UCx
.....
UCy
Use cases
Describes functional aspects, actors and workflows
REQx
.....
Requirements
Describes functional and non-functional requirements
Where are the conceptual gaps = missing mappings ?
CAPx
.....
Capabilities
Describes existing and future capabilities
...(T1 OR T2) AND T3...
Where are the technology gaps = missing mappings ?
What are the technology options = multiple mappings ?
TECHx
.....
Technologies
Describes existing and future technologies incl. products available on the market

Figure 2. The five artefact levels of the SERVUS methodology.

In addition, the relationships between these artefacts are specified, e.g., user stories motivate use cases, use cases are mapped to requirements, requirements may be fulfilled by capabilities, capabilities are realizing requirements. This guarantees a full bijective traceability of the artefacts, which helps setting development priorities and deriving implementation roadmaps out of the capabilities.

An important aspect for the common understanding of all the artefacts is the agreement upon the terms that are used by both the end users and the software architects. Very often, these terms are captured during the analysis phase in a glossary. The SERVUS methodology foresees linking the terms used in the SERVUS elements to concepts described in a glossary or formally defined in an ontology (semantic annotation).

3.1.2. Relations between the SERVUS Meta-Model Elements

There are the relations between the SERVUS meta-model element types: user stories (US), use cases (UC) are linked to actors, to other use cases, to test cases (TC), to requirements (REQ), and information resources (IR). These relations are illustrated in Figure 3 and are explained below.

- US to Actor
  - tells (inverse relation: is told by): a US is told from the viewpoint of an actor.
- US to UC
  - motivates (inverse relation: is motivated by): a US motivates the description of functional requirements in the form of use cases (UC).
- UC to Actor
  - performs (inverse relation: is performed by): a UC is initiated and performed by an actor.
- UC to UC
  - includes (inverse relation: is included in): one UC is included in another UC, i.e., one UC is included as a whole in the main success scenario, extension, or alternate path of another UC.
  - refines (inverse relation: abstracted from): one UC is a refinement of another UC, e.g., it provides more details in its main success scenario, adds an extension or interprets a more abstract UC in the context of a thematic domain.
- UC to TC
  - is tested by (inverse relation: tests): one UC may be tested by one or more test cases. One test case may also be linked to several use cases (which are then included in each other).
- UC to REQ
  - maps to (inverse relation: is derived from): a UC is mapped to one or more requirements. This means that the system under design should provide function (may be in terms of a Web service) that fulfills each requirement.
- REQ to REQ
  - related to (bijective relation): one REQ is related to another REQ, i.e., there is some relationship between the requirements. This relation has to be qualified in the comments. It could be a unilateral or bilateral dependency but also some similarity in terms of concepts, design pattern or technology.
- REQ to CAP
  - implemented by (inverse relation: implements): one REQ is implemented by one or more CAPs. This relation has to be better qualified in the future. It could be a unilateral or bilateral dependency but also some similarity in terms of concepts, design pattern, or technology.

- UC to IR
    - requests (inverse relation: is requested by): a UC requests an information resource in a defined access mode (create, read, update, delete).
- CAP to TECH
    - realized by (inverse relation: realizes): a capability is realized by a combination of technologies whereby the combination may be specified by simple Boolean expression.
- TECH to TECH
    - belongs to (inverse relation: comprises): a technology/product may belong to another technology/products. This relationship allows the user to express structured technologies/products.
- IR to IR
    - refines (inverse relation: abstracted from): an information resource is a refinement of another information resource (in the sense of inheriting all properties of the more abstract information resource).
    - related to (bijective relation): an information resource is related to another information resource. The meaning of the relation may be defined during the information modelling design step.

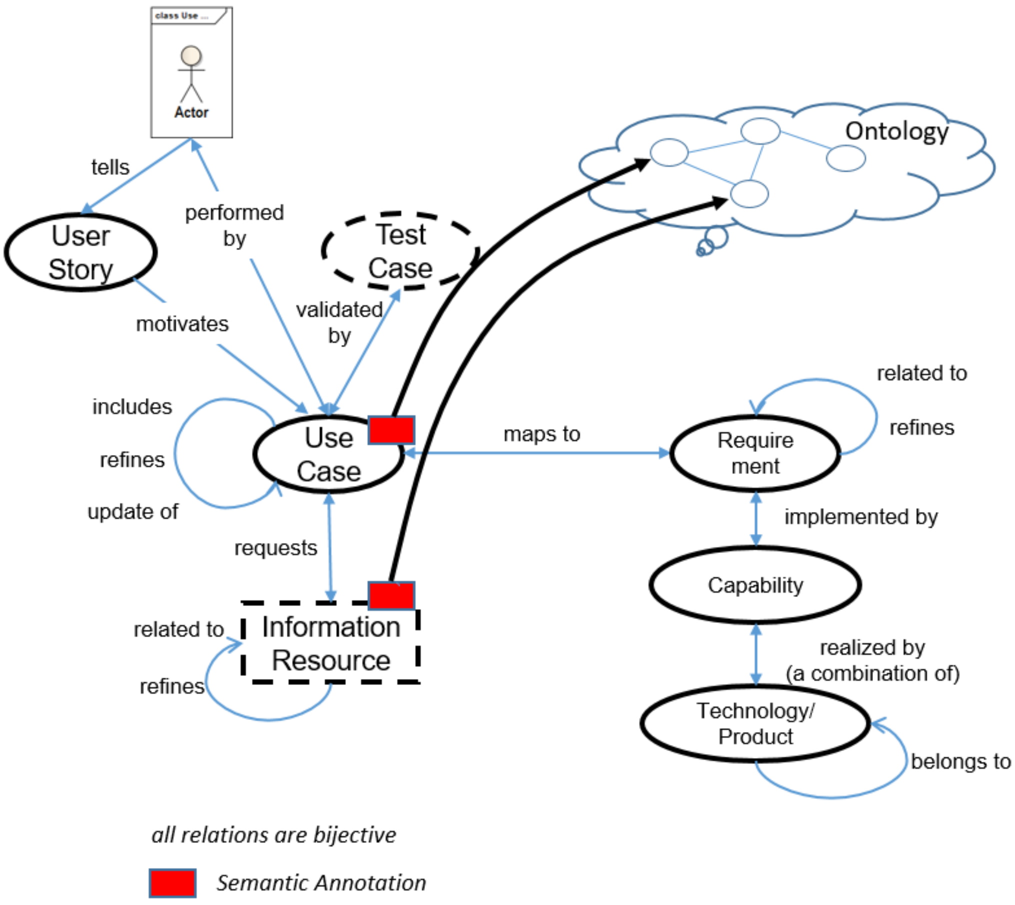

Figure 3. Relations between the major SERVUS meta-model elements.

An important aspect of the requirements and system analysis phase is the agreement upon the terms that are being used by both the end users and the software architects. Very often, these terms are being captured during the analysis phase in a glossary. The SERVUS methodology foresees linking the terms being used in the SERVUS elements to concepts described in a glossary, or formally defined in an ontology. The linkage is also called semantic annotation.

3.2. D2: Analysis and Design Activities

The following SERVUS analysis and design activities are required in order to specify the artefacts [4]:

- Domain modelling: defines the basic concepts of the thematic domain to which the problem belongs, and their interrelationships. Usually, this activity is carried out by experts of professional organizations representing a thematic community or outstanding institutions such as universities or research institutes.
- Problem analysis: derives the set of functional, informational, and qualitative requirement from the problem to be solved and documents them in natural language in electronic format (marked as "req's" in Figure 4).
- Feedback generation (optional): re-formulates the formal specification of the capability into the original language of the user and explains how the original user requirements have been satisfied.
- Rephrasing: translates and relates the artefacts of the requirements to the concepts of a design model.
- Publishing: represents the step in which the capabilities of the selected platform are entered into the capability model as part of the design model.
- Grounding: maps the (new or extended) capabilities to the specification style and language of the IIoT service platform and adds it to the set of platform capabilities (marked as "cap's" in Figure 4). The grounding activity is usually supported by engineering tools.
- Discovery: searches for capabilities that are candidates to fulfill the requirements.
- Matchmaking: maps requirements with capabilities. It comprises the evaluation of the adequacy of the candidate capabilities (i.e. types or instances) proposed by the discovery activity, the selection of one or more candidate capabilities, and finally the documentation of the mapping in the design model for traceability (marked as "req2cap" model in Figure 4).

Each element type has its own structure and template, i.e., its own set of text elements. If requirements analysis and system analysis have been performed according to this meta-model, i.e., the artefacts of each level are entered in the Platform Engineering Information System as described in Section 5, it is possible to exploit the system

- to find conceptual gaps on CAP level, and therefore, by variation and addition of further user stories and use cases, to work towards a specification of an "ideal" platform,
- to identify technology gaps, i.e., capabilities that may not (yet) be realized, and
- to identify technology options, i.e., capabilities that may be realized by a multitude of technologies and their combinations.

For completeness, please note that SERVUS methodology comprises two further element types that are, however, not (yet) used in the PEIS:

- Test cases: describes a possible instantiation of a use case that is decisive for the system test with respect to this use case.
- Information resources: describes the information elements including its basic operations (create, read, update, delete) that are required to carry out the use case (following the resource-oriented approach of SERVUS).

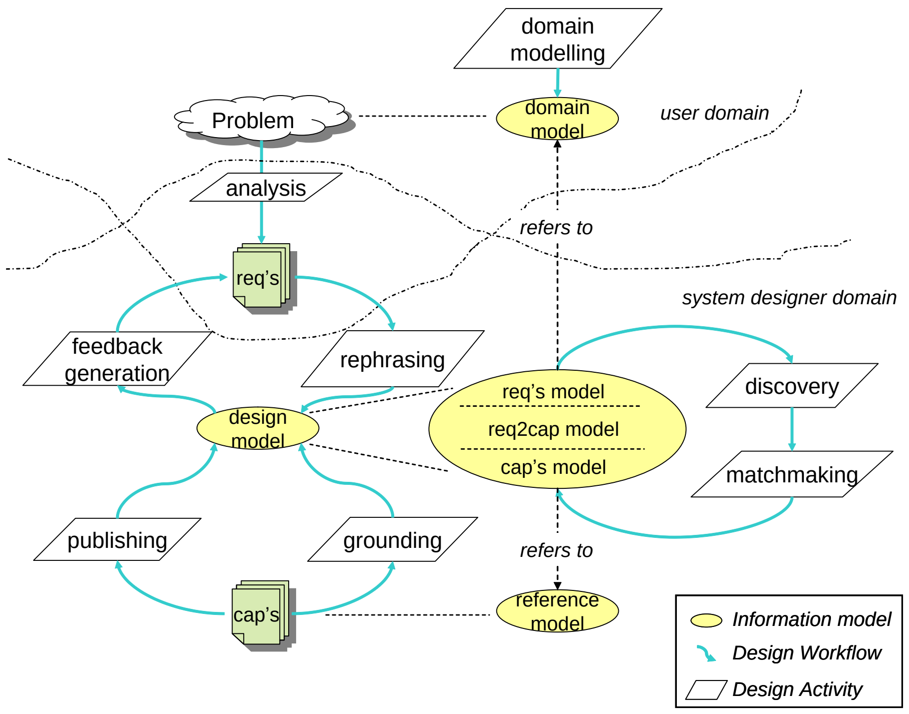

Figure 4. SERVUS analysis and design activities [4].

3.3. D3: Relationship to Reference Architecture Models

The third dimension is related to the positioning of the analysis and design artefacts and activities with respect to (standardized) architecture reference models. In the case of the IIoT and Industrie 4.0, this is related to the Industrial Internet Reference Architecture specified by the Industrial Internet Consortium (IIC) [2] and the Reference Architecture Model Industrie 4.0 (Figure 5) [3]. Referring to the three dimensions of the SERVUS methodology (see Figure 1), these architectural aspects belong to the dimension D3 "architectural frameworks".

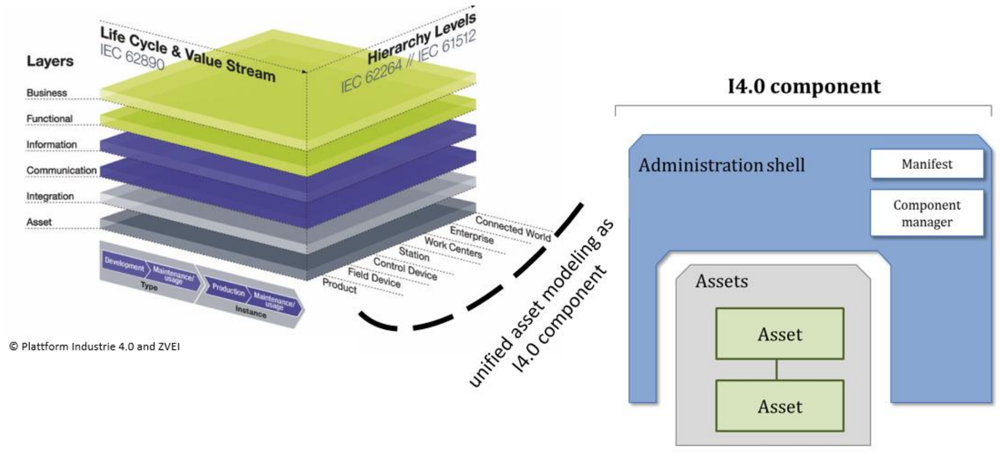

Figure 5. Main Industrie 4.0 concepts: RAMI4.0 and I4.0 component (graphics of the figure taken from Reference [3]).

The SERVUS requirements analysis approach refers to all architectural aspects of RAMI4.0 (referred to as "layers"). The artefacts "user story" and "use case" may be positioned in the RAMI4.0 business layer whereas they contain references to all other layers too. It encompasses all assets encapsulated in I4.0 components focusing on the functional interface of its administration shell and the meta-data ("manifest") of the I4.0 component.

The SERVUS requirements contain references to the asset types in the RAMI4.0 hierarchy level dimension (e.g., cloud-based data storage, controller device, production cell) upon which the requirement is set.

The IIRA defines five so-called functional domains, which provide the major building blocks for Industrial Internet Systems (see Figure 6):

- Control domain
- Operations domain
- Information domain
- Application domain
- Business domain

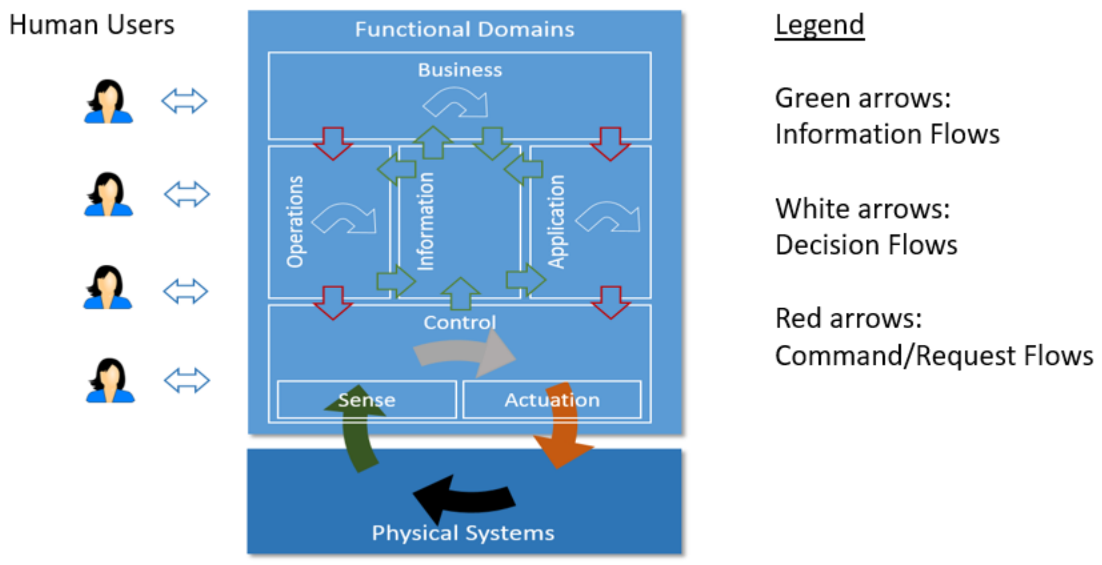

Figure 6. The Industrial Internet Reference Architecture (IIRA) functional domains [2].

These functional domains may be refined according to the needs of a project and an application domain, and mapped to architectural patterns, e.g., the three-tier architecture pattern as illustrated in Figure 7.

The SERVUS methodology supports this refinement. It foresees defining lists of project-specific "topics" and assigning them to IIRA functional domains. Requirement and capabilities artefacts may also refer to these topics.

Figure 8 shows an example of a landscape for the capability development in relation to the Architecture Elements (adopted from and positioned according to the three-tier IIS Architecture in IIRA). The colored circles ("dots") indicate how many capabilities have been identified for the respective Architecture Element at the given time horizon up to 2025. Note that a capability may relate to several Architecture Elements. In fact, there are 114 relations of a capability to an Architecture Element in Figure 8, but only 57 distinct capabilities. The area of each dot is proportional to the number of respective capabilities. The green dots for 2015 refer to existing capabilities, the others to planned capabilities. Such reports help to the project manager to plan the activities for the upcoming years and assign corresponding personal resources.

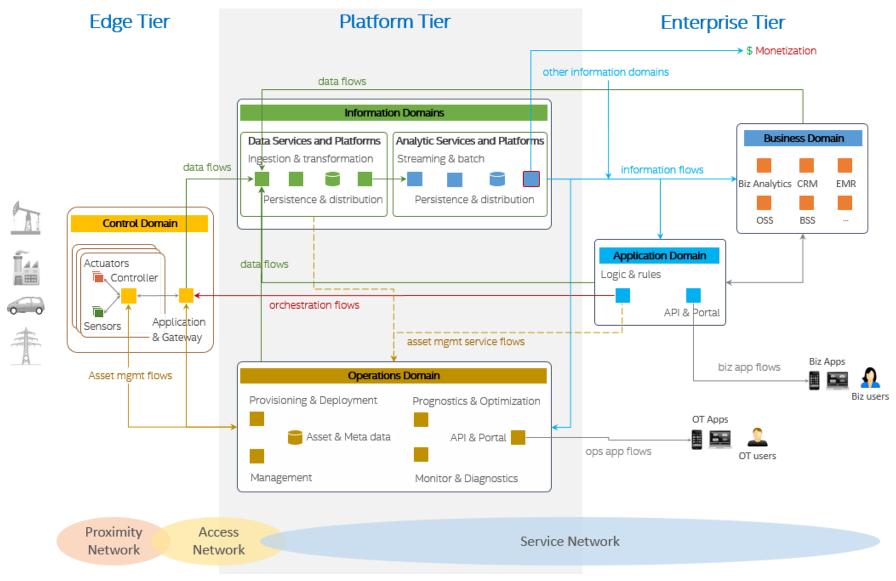

Figure 7. IIRA three-tier architectural pattern [2].

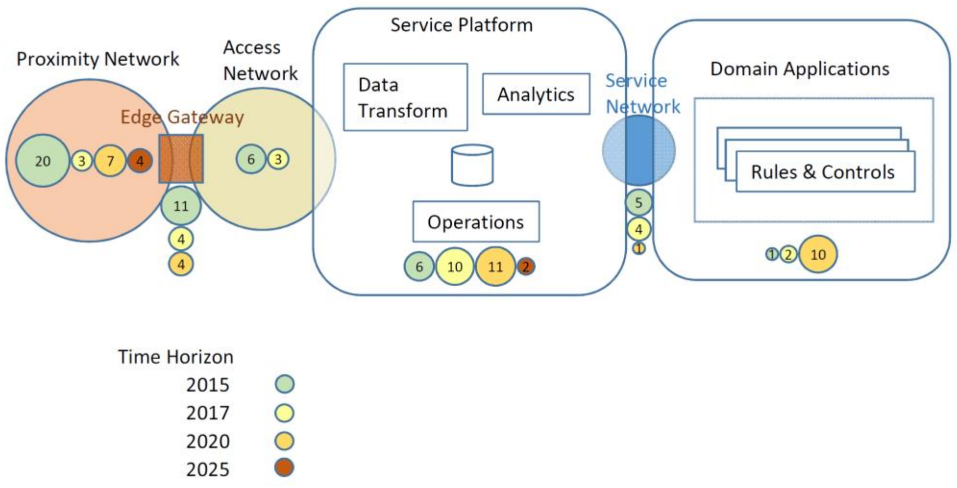

Figure 8. Landscape of Capabilities in relation to the architecture elements.

4. Platform Engineering Information System (PEIS)

4.1. Tool Description

Non-trivial projects and related system analysis and design activities require a tool that supports the edition and documentation of the use cases. The SERVUS design methodology is supported by a dedicated Web-based Platform Engineering Information Systems (PEIS), which allows the users to work in a collaborative, distributed and agile manner.

The PEIS starts in any desired Web-Browser with a dashboard-like home screen (Figure 9).

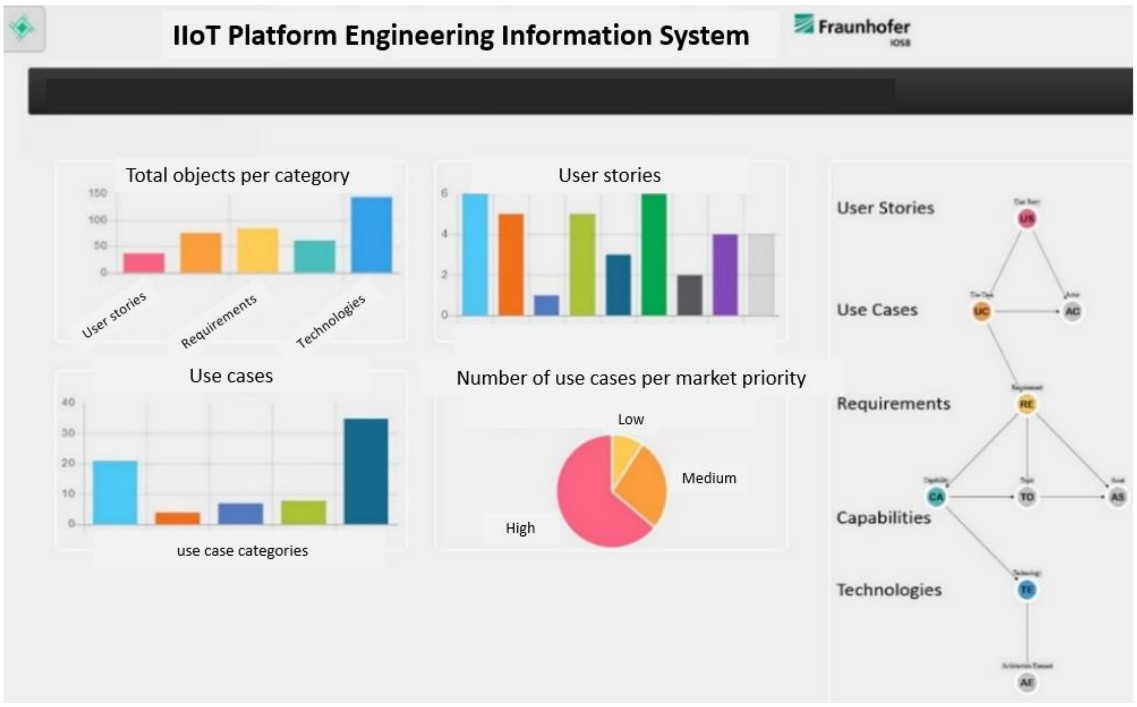

Figure 9. Dashboard of the IIoT Platform Engineering Information System (PEIS).

The home screen includes a main menu in black and an address link that can also be used for navigation through the server content, as well as an area with diagrams and graphs below, which offers the server content graphically. The following diagrams are located on the home screen:

- Total number of objects per category e. g., use cases, capabilities, etc.
- Number of user stories per user story group e.g., Flexible Manufacturing (FM), Human centered manufacturing (HCM), etc.
- Number of use cases per use case group e.g., Plant construction and operation (PCO), Process and production line development (PR), etc.
- Number of use cases per market priority as a pie chart with the segments representing high, medium, and low market priority.
- On the right hand side a graph representing the meta-model of the server content. With the concepts e.g., user story, use case, actor, etc., as nodes and the relations between the concepts as edges e.g., "use case has actor".

These diagrams may be used as entry points to query the PEIS and get detailed information about the SERVUS meta-model elements such as user stores, use cases, requirements, capabilities, and technologies. Each instance has a unique identifier so that it may be referenced in reports and discussions. The relationships between these elements are realized by hyperlinks. At each point in time a report may be generated from the PEIS contents and delivered as document in PDF format.

The PEIS architecture relies upon a Web-based collaborative content, community, and knowledge management framework called WebGenesis®. Based upon an ontology-based workflow engine and graphical user interface [17], it allows a software engineer to quickly adapt form-based content acquisition, editing workflows, and layout according to the needs of the software engineering projects supported and accompanied. Using Web-based technology, it enables collaborative work in a distributed manner, taking into account locations of software engineering teams dispersed around the world. A direct link to UML tools and frameworks is possible such that UML models (use case models, activity diagrams, etc.) may be attached to the analysis and design artefacts.

The current work focuses on the integration of data analysis capabilities such as machine learning and reporting capabilities to support decision making in systems and service engineering for IIoT applications following the latest architectural frameworks and technologies in Industrie 4.0 and beyond.

4.2. Content Generation

Following the SERVUS methodology and the usage of the PEIS the content generation process shall follow two perspectives and activities, also personalized by two independently working teams:

- a top-down approach that comprises the requirements analysis driven by the market demands. It encompasses the contents generation of the meta-model elements user story (US), use case (UC), and requirement (REQ). The requirement analysis shall be performed by those people that know the market and/or have a good understanding of the user demands. For the application domain of industrial production, the requirements analysis team may comprise industrial engineers, production planners, or factory automation experts of the higher levels of the automation pyramid.
- a bottom-up approach that comprises the technology analysis driven by the ongoing and rapid evolution of the automation technology (also called operational technology – OT) and the information technology (IT). It encompasses the contents generation of the meta-model elements technologies/products (TECH) and the capabilities (CAP). For the application domain of industrial production, the technology analysis team may comprise computer scientists, data scientists, factory automation experts of the lower levels of the automation pyramid, e.g., for (real-time) control of automation processes.

These two activities may work largely independently and parallel to each other. Their analysis results (artefacts) shall be entered into the PEIS and finally result in a set of documented prioritized requirements (REQs) and documented capabilities (CAPs) including their availability time, both with their relationships to the other meta-model elements. Hence, traceability of both activities is given. Then, in a further step, the REQ2CAP mapping may take place. This mapping activity is crucial as:

- it leads to the identification of the conceptual gaps between the REQs and the CAPs, and therefore
- is an essential indication of the developments to be undertaken in order to close the gaps.

The strategic mapping activity needs facilitation by an experienced senior manager who is experienced in both requirements analysis and OT/IT technologies. However, this activity needs guidance. This guidance is given by priorities to be set by the product managers and the software engineers who both have to take responsibility for the software platform development process and the delivery of its result in time and good quality.

Experience shows that the overall process cannot be concluded in a single step activity because of its inherent complexity and the changing conditions and influencing factors over time. The solution to this problem is an agile approach, i.e.,

- the requirements analysis activity needs an adjustment, re-structuring, re-phrasing, and re-linking during and after the strategic mapping in order to clarify issues and include new insights from a changing market development and analysis,
- the technology analysis activity needs an adjustment, re-structuring, re-phrasing, and re-linking during and after the strategic mapping in order to clarify issues and include new insights from a changing technology (OT and IT) market development and analysis.

As a result of such activities, the REQ2CAP mapping may also need a re-adjustment, leading to an iterative REQ2CAP mapping. However, it is important to fix the content from time to time for milestones to be achieved in the development. This is similar to sprints in the scrum methodology for software development. Figure 10 illustrates the agile content generation process based upon the SERVUS methodology when being applied to IIoT platform engineering.

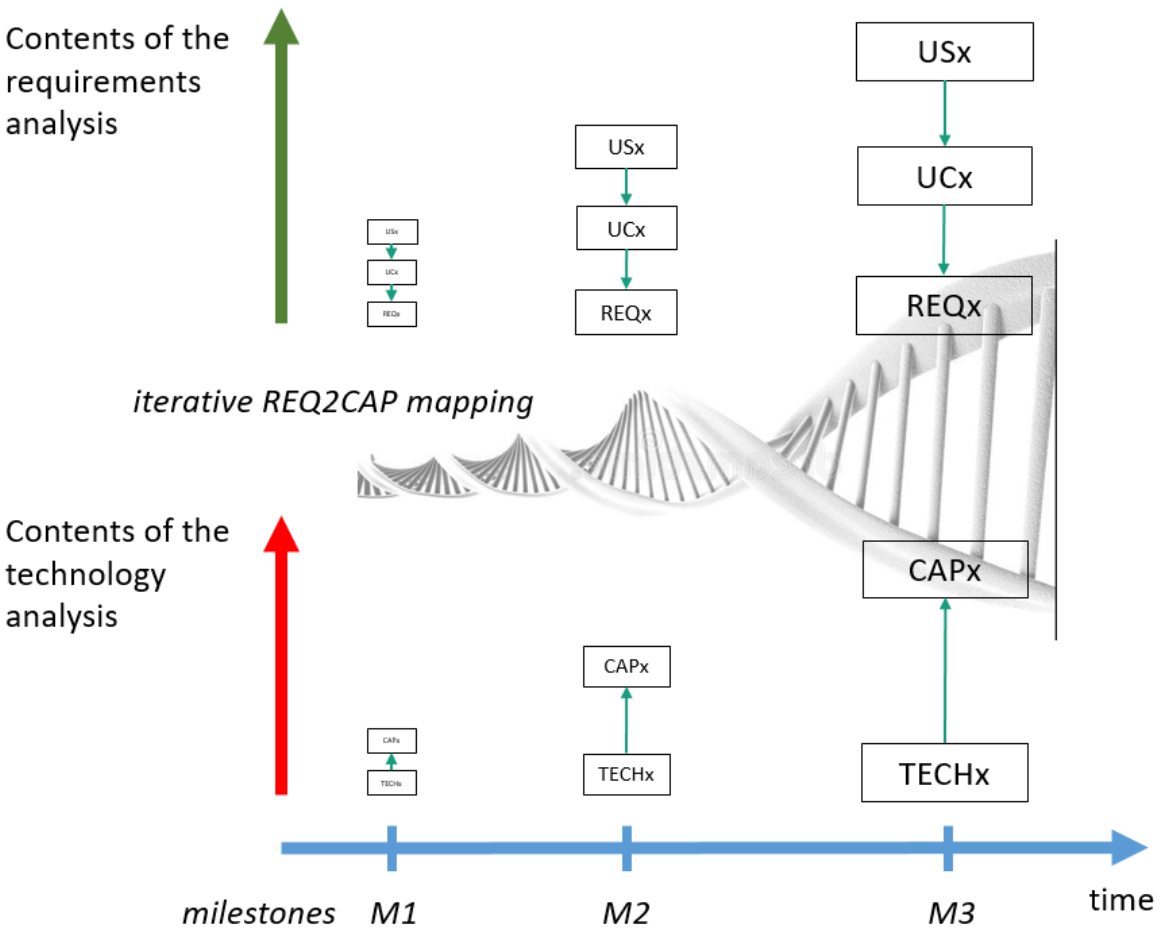
Contents of the requirements analysis
USx
UCx
REQx
iterative REQ2CAP mapping
Contents of the technology analysis
CAPx
TECHx
milestones
M1
M2
M3
time

Figure 10. Agile content generation for the PEIS and iterative REQ2CAP mapping.

5. Discussion

All these activities are interconnected by a common modelling environment, which is structured according to the SERVUS meta-model and supported by a Platform Engineering Information System (PEIS). These activities enable and support a co-design of these design artefacts [19] in an iterative and agile manner such that the design solution may be elaborated and traced back step-by-step.

5.1. Agile Modelling Best Practices

These activities follow the 14 best-practice principles of Agile Model Driven Development (AMDD) (Table 1) [20].

Table 1. Application of Agile Modelling Best Practices in SERVUS.

No.	Agile Modelling Best Practices	Application in SERVUS
1	**Active stakeholder participation.** Stakeholders should provide information in a timely manner, make decisions in a timely manner, and be as actively involved in the development process through the use of inclusive tools and techniques.	Actors represent the stakeholders in the use case modelling.
2	**Architecture envisioning.** At the beginning of an agile project you will need to do some initial, high-level architectural modeling to identify a viable technical strategy for your solution.	The models shall be drafted according to reference models and architectural frameworks (see SERVUS dimension D3). A co-design of requirements and architectural artefacts is possible.
3	**Document Continuously.** Write deliverable documentation throughout the lifecycle in parallel to the creation of the rest of the solution.	The continuous documentation is enabled and enforced by the Web-based Platform Engineering Information System (PEIS) [1].
4	**Document Late.** Write deliverable documentation as late as possible, avoiding speculative ideas that are likely to change in favor of stable information.	A deliverable may be generated automatically from the PEIS all the time, i.e., also at given milestones in the project when a certain level of stability is assumed. No further and extra effort is necessary for documentation purposes.
5	**Executable Specifications.** Specify requirements in the form of executable "customer tests", and your design as executable developer tests, instead of non-executable "static" documentation.	Each entry in the PEIS may be linked to Unified Modeling Language (UML) model elements that may be used as starting points to code generation.
6	**Iteration modelling.** At the beginning of each iteration, a bit of modelling is done as part of the iteration planning activities.	The design model may be adapted whenever required.
7	**Just barely good enough (JBGE) artefacts.** A model or document needs to be sufficient for the situation at hand and no more.	The "design model" can evolve according to the needs. This allows analysts, architects, and designers to tailor and minimize the effort to be put into the documentation of the artefacts according to the needs and the resources available.
8	**Look-ahead modelling.** Sometimes requirements that are nearing the top of your priority stack are fairly complex, motivating you to invest some effort to explore them before they're popped off the top of the work item stack so as to reduce overall risk.	The "design model" can evolve according to the needs. Each requirement and each capability can be given a priority.
9	**Model storming.** Throughout an iteration a brainstorming session can be held, called "model storm" on a just-in-time basis for a few minutes to explore the details behind a requirement or to think through a design issue.	The PEIS is Web-based such that brainstorming sessions may even be carried out in distributed or virtual organizations spread over several locations.
10	**Multiple models.** Each type of model has its strengths and weaknesses. An effective developer will need a range of models in their intellectual toolkit enabling them to apply the right model in the most appropriate manner for the situation at hand.	The design model can be documented in several forms: semi-structured table, UML model or a figure.
11	**Prioritized requirements.** Agile teams implement requirements in priority order, as defined by their stakeholders, so as to provide the greatest return on investment possible.	Requirements have a priority field, hence the work may be structured and planned according to the priorities.
12	**Requirements envisioning.** At the beginning of an agile project, you will need to invest some time to identify the scope of the project and to create the initial prioritized stack of requirements.	The scope and granularity of use cases and requirements may be tailored according to the knowledge that is existing at a certain point in time. Both use cases and requirements may be related to each other and refined as required.
13	**Single source information.** Strive to capture information in one place and one place only.	The PEIS is a Web-based collaborative tool, i.e., all users are working on a single project instance of server and have the same visibility according to their roles.
14	**Test-driven design (TDD).** Write a single test, either at the requirements or design level, and then just enough code to fulfill that test. TDD is a just-in-time approach to detailed requirements specification and a confirmatory approach to testing.	Test cases may be added as an optional further artefact to SERVUS.

[1] See Section 4 describing the Platform Engineering Information System (PEIS).

5.2. Evaluation

Originally, the SERVUS methodology and the accompanying PEIS were designed for environmental and risk management projects where the IIoT platform comprises the set of available standards and technologies of the geospatial communities [4,15]. With the ongoing digitization of the industrial production due to initiatives such as Industrie 4.0, SERVUS was applied in IIoT platform development projects with industrial partners. Here, future-oriented hypothetical use cases had to be drafted, broken down to platform requirements, prioritized, and matched with already existing platform capabilities such that the resulting commercial platform is competitive for the upcoming years and future use cases of customers.

Experience in these projects shows that the SERVUS methodology basically satisfies the needs and the expectations of the project and development managers and supports their planning and decision-making. Criticism is expressed with respect to the required effort. On the one hand, considerable effort has to be spent to keep the contents of the PEIS consistent and up-to-date. On the other hand, as the PEIS provides information retrieval and dashboard functionalities, it plays the role of an active tool and utility instead of huge amounts of passive formal or textual documents. As SERVUS and PEIS only provide a modelling framework with a multitude of optional elements, the effort may be adjusted according to the project constraints.

It may be desirable to try to evaluate the advantages and disadvantages of software engineering methodologies as well as to quantify the efficiency gains in real-world settings. In experimental environments with a manageable complexity, a setting may be defined with different project teams working in parallel but following different software methodologies. Here, an objective comparison may be possible. However, in a world of agile engineering approaches, where incremental solutions are provided step-by-step in close collaboration with the users, and in our context, the platform providers, such a setting is not possible. This is also the reason why, although such empirical research is demanded in recent literature [8], no results are reported. Furthermore, the question is raised as to whether there is a need at all for a single method for the Internet of Things [21].

Personal experience with the SERVUS methodology shows that, at the beginning of a software engineering project in an IIoT context, members of projects teams are, at first glance, reluctant to use the methodology because of its apparent complexity. Project members think they do not need such a methodology and start with classical methodologies such as the UML-based Rational Unified Process (RUP). However, after some analysis steps, it turns out that commonalities among use cases and requirements are detected leading to resource-expensive and time-consuming reorganizations of the analysis and design artefacts. Furthermore, it becomes clear that some of the requirements are already fulfilled by capabilities in IIoT platforms. Very quickly, requests with regard to describing such artefacts in a way that information retrieval is possible appear, which then lead to the decision that a better fitting approach, such as SERVUS and its associated PEIS tool, is required that allows the interlinking of artifacts. Despite the wish for an engineering information system, the possibility to always be able to generate a (theoretically) printable document (in a PDF format) is highly appreciated.

6. Conclusions

The SERVUS methodology presented in this article tackles the problem of service engineering in an IIoT context. Its basic characteristic is the interlinked and agile co-design of requirements and capabilities artefacts resulting from:

1. a user point of view in a top-down approach from user stories, use cases to requirements, and
2. a platform point of view in a bottom-up approach encompassing existing and emerging technologies and products that are used to provide specified capabilities of an IIoT platform.

On the basis of these analysis and design activities it is the task of the system architect and designer to map the requirements artefacts to the capabilities artefacts and derive from this the development roadmap. Hence, the originality of the SERVUS methodology lies in the integrated comprehensive offering of the following features:

1. SERVUS combines a top-down with a bottom-up approach in a systematic manner.
2. SERVUS relies upon a common language (semi-formal textual descriptions with optional formal models as attachments) for all analysis and design artefacts such that the cultural and language gap between thematic experts and IT experts may be overcome.
3. SERVUS is accompanied by a Web-based collaborative Platform Engineering Information Systems (PEIS) that supports the online documentation of all analysis and design activities.
4. SERVUS allows an agile engineering process with refinements and extensions that follow the priorities and the progress of the project in short development cycles.
5. SERVUS supports the 14 best-practice principles of Agile Model Driven Development.
6. SERVUS distinguishes between a conceptual, technology-independent level, and technology-specific descriptions.

Future work will focus on data analysis tools to validate and visualize the contents of the PEIS in order to automatically detect inconsistencies. Special attention will be dedicated to describing requirements and capabilities compliant to the emerging detailed specifications of an Asset Administration Shell of the Industrie 4.0 initiative [3]. This would lead to turn the matchmaking of requirements with capabilities into a matchmaking between I4.0 component specifications following the idea of interacting I4.0 Components in an Interaction-based Architecture of Industrie 4.0 [22]. Furthermore, we are working with the objective to provide distinct project spaces for customer projects, especially for the user requirements artefacts, such that the contents of the IIoT capabilities and technology artefacts reflect the characteristics of existing IIoT platforms and may be reused across customer project spaces.

Author Contributions: The work presented in this paper represents a collaborative effort by both authors, whereas T.U. wrote the main paper and contributed the SERVUS Methodology based on his doctoral thesis [4]. T.B. validated the methodology and tailored it to the application domain of integrated environmental information and risk management systems. Both authors have read and approved the final manuscript.

Funding: This research received no external funding by a public agency.

Acknowledgments: This research relies upon multiple projects and activities within the department Information Management and Production Control (ILT) of Fraunhofer IOSB thanks to manifold contributions of many research scientists of this department, especially for the PEIS development.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Al-Gumaei, K.; Schuba, K.; Friesen, A.; Heymann, S.; Pieper, C.; Pethig, F.; Schriegel, S. A Survey of Internet of Things and Big Data Integrated Solutions for Industrie 4.0. In Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Torino, Italy, 4–7 September 2018.
2. Industrial Internet Consortium (IIC). The Industrial Internet Reference Architecture. Technical Article. Available online: http://www.iiconsortium.org/IIRA.htm (accessed on 29 August 2018).
3. *DIN SPEC 91345. Reference Architecture Model Industrie 4.0 (RAMI4.0)*; Beuth-Verlag: Berlin, Germany, 2016.
4. Usländer, T. Service-oriented Design of Environmental Information Systems. PhD Thesis, Karlsruhe Institute of Technology (KIT), KIT Scientific Publishing, Karlsruhe, Germany, 2010.
5. Jacobson, I.; Ng, P.-W. *Aspect-Oriented Software Development with Use Cases*; Addison-Wesley: Boston, MA, USA, 2005.
6. Cockburn, A. *Writing Effective Use Cases*; Addison-Wesley: Boston, MA, USA, 2001.
7. Kohlborn, T.; Korthaus, A.; Chan, T.; Rosemann, M. Service Analysis—A Critical Assessment of the State of the Art. In *Proceedings of the European Conference of Information Systems (ECIS 2009), Verona, Italy, 8–10 June 2009*; Association for Information Systems: Atlanta, GA, USA, 2009.

8. Reyes-Delgado, P.Y.; Mora, M.; Duran-Limon, H.A.; Rodríguez-Martínez, L.C.; O'Connor, R.V.; Mendoza-Gonzalez, R. The strengths and weaknesses of software architecture design in the RUP, MSF, MBASE and RUP-SOA methodologies: A conceptual review. In *Computer Standards & Interfaces 47*; Elsevier: Amsterdam, The Netherlands, 2016; pp. 24–41.
9. Lutz, M. Ontology-based Descriptions for Semantic Discovery and Composition of Geoprocessing Services. *Geoinformatica* **2007**, *11*, 1–36. [CrossRef]
10. Blomstedt, F.; Ferreira, L.L.; Klisics, M.; Chrysoulas, C.; de Soria, I.M.; Morin, B.; Zabasta, A.; Eliasson, J.; Johansson, M.; Varga, P. The Arrowhead Approach for SOA Application Development and Documentation. In Proceedings of the IEEE IECON 2014, Dallas, TX, USA, 30 October–1 November 2014; pp. 2631–2637.
11. Varga, P.; Blomstedt, F.; Ferreira, L.L.; Eliasson, J.; Johansson, M.; Delsing, J.; de Soria, I.M. Making System of Systems Interoperable—The Core Components of the Arrowhead Framework. *J. Netw. Comput. Appl.* **2017**, *81*, 85–95. [CrossRef]
12. Usländer, T. Agile Service-oriented Analysis and Design of Industrial Internet Applications. In *Factories of the Future in the digital environment—Proceedings of the 49th CIRP Conference on Manufacturing Systems*; Westkämper, E., Bauernhansl, T., Eds.; Elsevier: Amsterdam, The Netherlands, 2016; Volume 57, pp. 219–223. Available online: https://www.sciencedirect.com/journal/procedia-cirp/vol/57/suppl/C (accessed on 29 August 2018).
13. Usländer, T.; Epple, U. Reference model of Industrie 4.0 service architectures: Basic concepts and approach. In *Automatisierungstechnik: AT 63 (2015), No.10*; Bretthauer, G., Ed.; de Gruyter Oldenbourg: Berlin, Germany, 2015; pp. 858–866.
14. IEC 62541 OPC Unified Architecture. Beuth-Verlag, 2016. Available online: https://webstore.iec.ch/publication/25997 (accessed on 8 October 2018).
15. Usländer, T.; Batz, T. How to Analyse User Requirements for Service-Oriented Environmental Information Systems. In Proceedings of the Environmental Software Systems. Frameworks of eEnvironment—9th IFIP WG 5.11 International Symposium, ISESS 2011, Brno, Czech Republic, 27–29 June 2011; Hrebícek, J., Schimak, G., Ralf Denzer, R., Eds.; Springer: Berlin, Germany, 2011; pp. 161–168.
16. ISO 19119:2016. Annex D: (informative) Use case-based methodology. In *ISO 19119 Geographic Information—Services*; ISO/TC 211 Geographic Information/Geomatics: Stockholm, Sweden, 2016.
17. Chaves, F.; Moßgraber, J.; Schenk, M.; Bügel, U. Semantic Registries for Heterogeneous Sensor Networks: Bridging the semantic gap for collaborative crises management. In *24rd International Conference on Database and Expert Systems Applications (DEXA 2013), Prague, Czech Republic, 26–29 August 2013*; Morvan, F., Ed.; IEEE Computer Society: Washington, DC, USA, 2013; pp. 118–122.
18. Van den Heuvel, W.J.; Zimmermann, O.; Leymann, F.; Lago, P.; Schieferdecker, I.; Zdun, U.; Avgeriou, P. Software Service Engineering: Tenets and Challenges. In *ICSE 2009 Workshop—Principles of Engineering Service Oriented Systems (PESOS), Vancouver, BC, Canada, 18–19 May 2009*; IEEE Computer Society: Washington, DC, USA, 2009.
19. Pohl, K.; Sikora, E. The Co-Development of System Requirements and Functional Architecture. In *Conceptual Modelling in Information Systems Engineering*; Krogstie, J., Opdahl, A.L., Brinkkemper, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 229–246.
20. Agile Modeling Core Practices. Available online: http://agilemodeling.com/essays/bestPractices.htmC (accessed on 29 August 2018).
21. Jacobson, I.; Spence, I.; Ng, P.-W. Is there a Single Method for the Internet of Things? *Commun. ACM* **2017**, *60*. [CrossRef]
22. Usländer, T. *DIN SPEC 16593-1 Reference Model for Industrie 4.0 Service Architectures—Part 1: Basic Concepts of an Interaction-based Architecture*; Beuth-Verlag: Berlin, Germany, 2018.