Co-Design of Requirements and Architectural Artefacts for Industrial Internet Applications

Conference Paper · October 2016

CITATIONS
0

READS
330

2 authors:

Thomas Usländer
Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IO…
98 PUBLICATIONS **507** CITATIONS
SEE PROFILE

Thomas Batz
Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IO…
13 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Co-Design of Requirements and Architectural Artefacts for Industrial Internet Applications

Thomas Usländer
Fraunhofer IOSB, Fraunhoferstr. 1, 76131 Karlsruhe. Email: thomas.uslaender@iosb.fraunhofer.de

Thomas Batz
Fraunhofer IOSB, Fraunhoferstr. 1, 76131 Karlsruhe. Email: thomas.batz@iosb.fraunhofer.de

Abstract: *The paper highlights the importance of systematic agile service engineering and a co-design of requirements and architectural artefacts for Industrial Internet and Industry 4.0 software applications. Here, the functional and non-functional requirements of IT users (mostly from the disciplines of mechanical engineering and electrical engineering) need to be mapped to the capabilities and architecture patterns of emerging Industrial Internet of Things (IIoT) service platforms. The capabilities of such platforms are usually described, structured and formalized by software architects. Hence, their 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 and the design of future platform capabilities. The paper describes how this 'gap' may be closed with help of a service-oriented development methodology entitled SERVUS and a corresponding Web-based collaborative documentation tool.*

Keywords: *Service-oriented analysis and design, Industrie 4.0, Industrial Internet of Things.*

Introduction

Today, in most cases, the development of industrial software applications cannot be isolated any more from the technological trends in the Internet and the more and more emerging Industrial Internet. This relates to the requirements of the users who request remote accessibility of data and services via the Internet to support new business models, as well as to the architecture, which need to be compliant to the current and future standards and products of the Industrial Internet of Things (IIoT). Analysis and design methodologies shall take into account this trend, too, when being applied in the IIoT context.

The Industrial Internet Reference Architecture (IIRA) (IIC, 2016) 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 applications domains, e.g. energy, healthcare, manufacturing, public sector, transportation and related industrial systems. The IIRA requests 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 (DIN, 2016).

Overview

The paper highlights the importance of systematic agile service engineering for Industrial Internet and Industrie 4.0 software applications. Here, the functional and non-functional requirements of IT users (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 (Usländer & Epple, 2015). 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 comprise the services of the OPC Unified Architecture. 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 paper describes how this 'gap' may be closed with help of a 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 (Usländer, 2010). SERVUS considers use case models as being core artefacts of requirement analysis.

Use case models describe the behavior of a system whereby according to Jacobson and Ng (2005) "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".

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 Cockburn's "Writing Effective Use Cases" book (Cockburn, 2001). This idea of a semi-structured description is also used when mapping the use cases to other design artefacts such as requirements and capabilities as described below.

The SERVUS methodology was successfully used in numerous collaborative and inter-disciplinary software projects related to various applications domains of the IIoT such as environmental information systems (Usländer, 2010), environmental risk management (Usländer and Batz, 2011) and industrial production/Industrie 4.0 (Usländer, 2016).

Methodology

The SERVUS methodology will presented according to its three dimensions (Fig. 1):

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

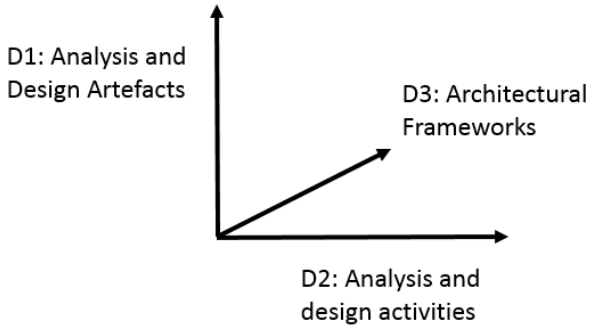


Fig. 1: Dimensions of the SERVUS Methodology

D1: Analysis and design artefacts

The paper describes the SERVUS meta-model of analysis and design artefacts. These encompass (see Fig. 2)

- **user stories** which describe selected situations in short textual snippets and motivate the further analysis process.

> Note: According to Wikipedia, a user story in software development and product management is a "description consisting of one or more sentences in the everyday or business language of the end user or user of a system that captures what a user does or needs to do as part of his or her job function. … It captures the

"who", "what" and "why" of a requirement in a simple, concise way, often limited in detail by what can be hand-written on a small paper notecard." For the SERVUS methodology, this basic idea is applied whereby the two main aspects are as follows: 1) user stories only imply weak requirements on conciseness and rigidity of the text and the terms used, and 2) they are captured in electronic form according to a semi-structured table such that textual search and retrieval as well as mapping to other analysis artefacts is possible.

- **use cases** which describe functional aspects, actors and workflows in a semi-structured manner.
- **requirements** which describe functional and non-functional needs in a semi-structured manner.
- **capabilities** which describe existing and future capabilities of an IIoT service platform.

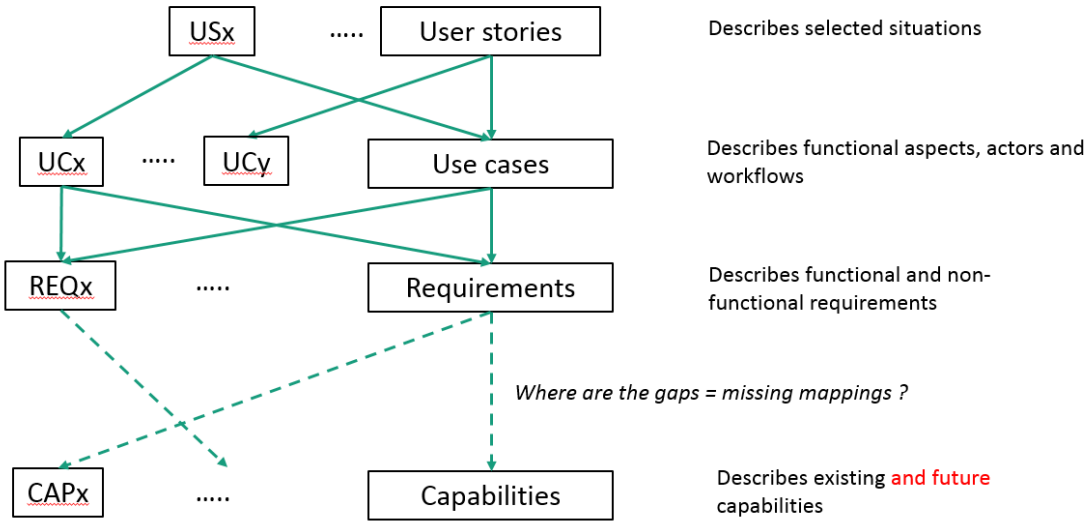


Fig. 2: The four artefact levels of the SERVUS Analysis and Design Methodology

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 to link the terms used in the SERVUS elements to concepts described in a glossary or formally defined in an ontology (semantic annotation).

D2: Analysis and design activities

The paper describes the SERVUS analysis and design activities that are required in order to specify the artefacts (Usländer (2010)):

- 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 document them in natural language in electronic format (marked as "req's" in Fig. 3).
- 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 Fig. 3). The grounding activity is usually supported by engineering tools.
- Discovery: searches for capabilities that are candidates to fulfil 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

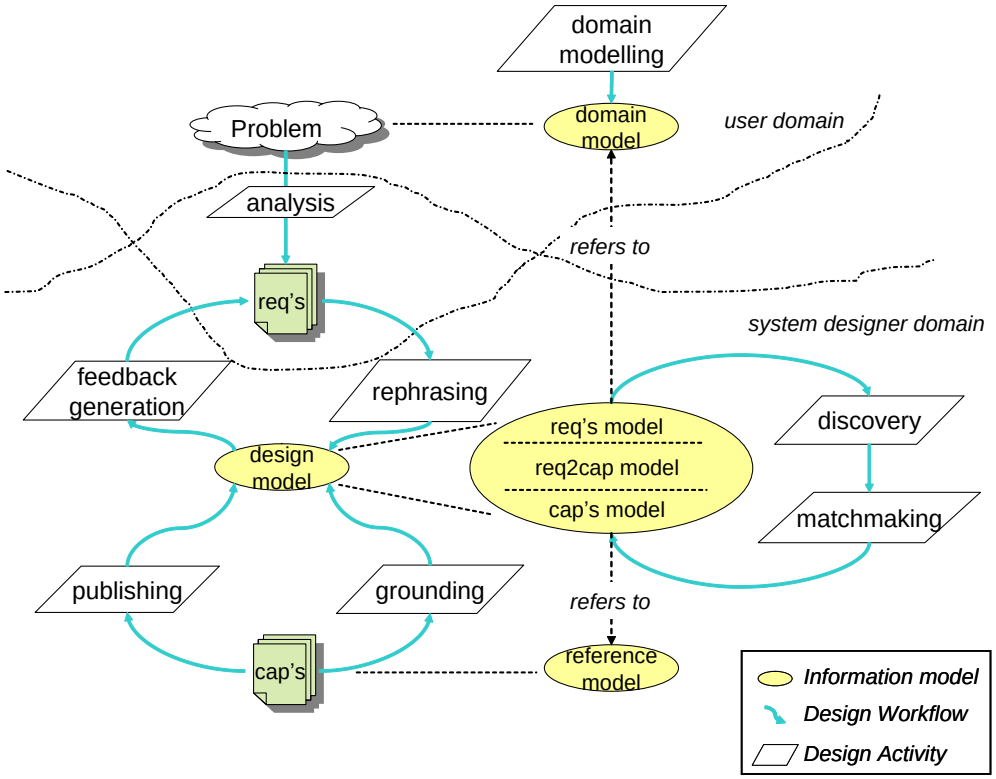


Fig. 3: SERVUS Analysis and Design Activities

All these activities are interconnected by a common modelling environment, which is structured according to the SERVUS meta-model. These activities enable and support a co-design of these design artefacts (Pohl & Sikora (2007) in an iterative and agile manner such that the design solution may be elaborated and traced back step-by-step.

These activities follow the best-practices principles of agile modelling (extracted from Wikipedia, 2016). Note, however, an agile software development methodology is not necessarily required.

Table 1: Application of Agile Modeling Best-Practices in SERVUS

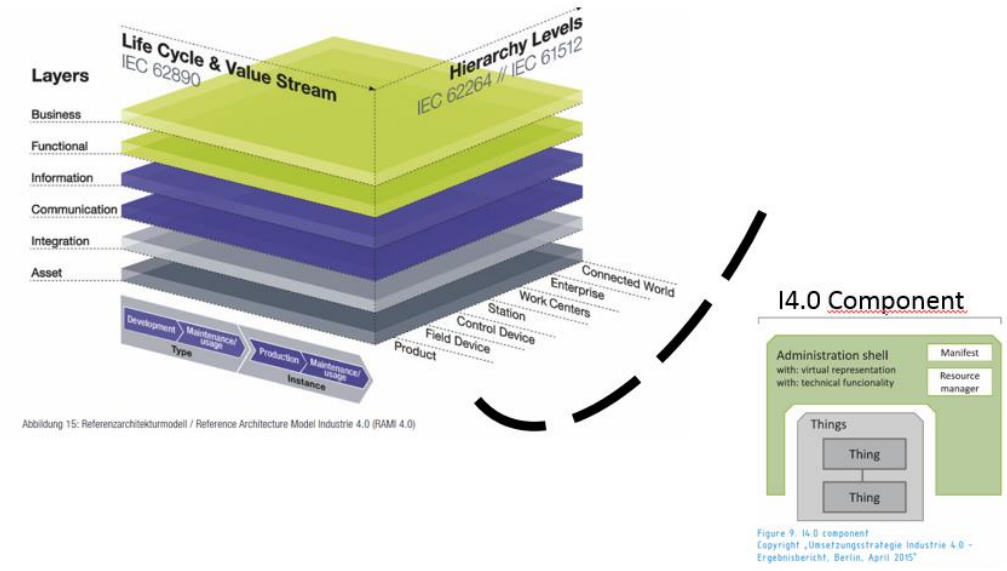
No	Agile Modelling Best-practice	Application in SERVUS
1	Just barely good enough artifacts. A model or document needs to be sufficient for the situation at hand and no more.	The "design model" can evolve according to the needs.
2	Architecture envisioning. At the beginning of an agile project, high-level architectural modeling is done to identify a viable technical strategy.	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	Lookahead modeling is used to reduce overall risk.	dito
4	Multiple models can be used. Each type of model has its strengths and weaknesses. Effective developers have 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.
5	Active stakeholder participation. Stakeholders are important for funding the process and accepting the results, that is why they are involved as soon as	Actors represent the stakeholders in the use case modelling.

	possible. Stakeholders provide information in a timely manner, make decisions in a timely manner, and are as actively involved in the development process as possible.	
6	Requirements envisioning. At the beginning of an agile project, time is invested 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.
7	Prioritized requirements. Requirements are implemented in priority order, as defined by their stakeholders, so as to provide the greatest return on investment possible. Collecting the low hanging fruit.	Requirements have a priority field.
8	Iteration modeling. At the beginning of each iteration, a bit of modeling is done as part of the iteration planning activities.	The design model may be adapted whenever required..
9	Test-driven development. Requirements are written like a test. Tests are performed and then just enough code is made to fulfill that test.	Test cases may be added as an optional further artefact to SERVUS.
10	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 SERVUS Tool (see below) is Web-based such that brainstorming sessions may even be carried out in distributed or virtual organisations spread over several locations.

D3: Relationship to reference architecture models (such as RAMI4.0 and IIRA)

The third dimension is related to the positioning of the analysis and design artefacts and activities w.r.t. to (standardized) architecture reference models. In the case of the IIoT and Industry 4.0 this related to the Industrial Internet Reference Architecture (IIRA) specified by the Industrial Internet Consortium (IIC) and the Reference Architecture Model Industrie 4.0 (RAMI4.0).

Looking at the SERVUS model categories (see Fig. **3**), these architectural aspects are related to the "reference model".



Plattform Industrie 4.0/Hrsg. BITKOM, VDMA, ZVEI:
Umsetzungsstrategie Industrie 4.0 – Ergebnisbericht, Berlin, April 2015

Fig. 4: Main Industrie 4.0 Concepts: RAMI4.0 and I40 Component

The SERVUS requirements analysis approach refers to all architectural aspects of RAMI4.0 (called “layers). The artefacts “user story” and “use case” may be positioned in the RAMI4.0 Business Layer whereas the 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 Fig. 5):

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

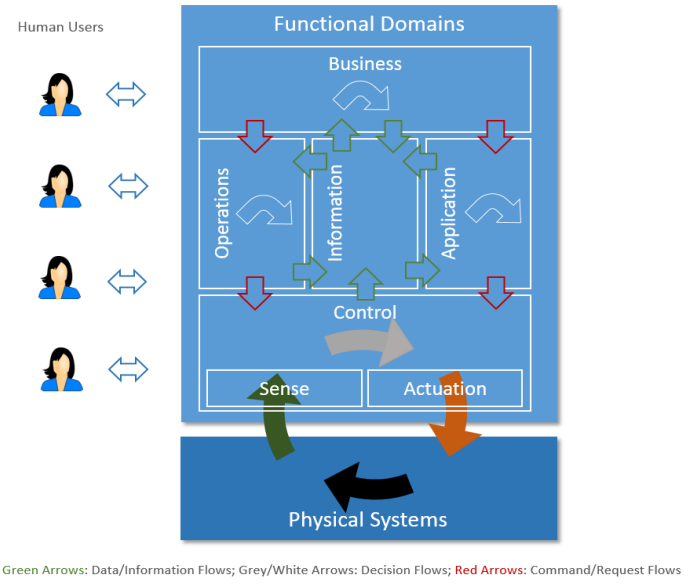


Fig. 5: IIRA Functional Domains

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 3-tier architecture pattern as illustrated in Fig. 6.

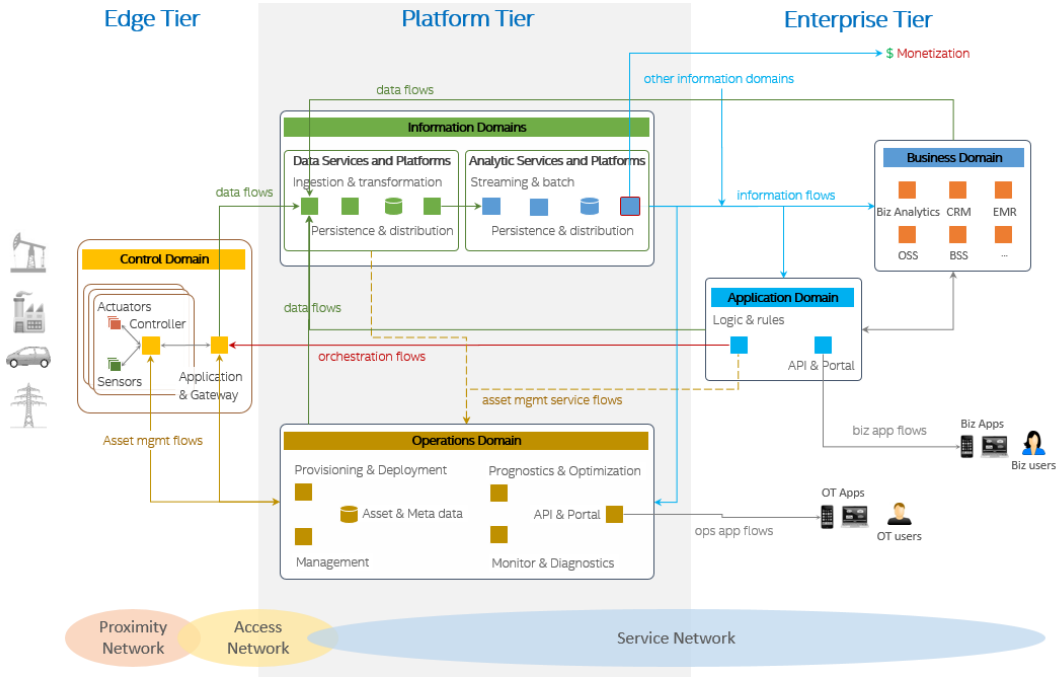


Fig. 6: IIRA Three-tier architectural pattern

The SERVUS methodology supports this refinement. It foresees to define lists of project-specific "topics" and assign them to IIRA functional domains. Requirement and capabilities artefacts may also refer to these topics. As a consequence, a landscape map of topics may be generated with an architectural pattern as a background. This helps to illustrate und understand the hot spots of the project development.

Tool Support – The SERVUS Use Case Server

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 tool, which allows the users to work in a collaborative and distributed manner. Furthermore, it supports an agile approach, i.e. use cases and the other elements in the model may be iteratively specified which allows the users to refine and change them according to the knowledge that is typically gained in the analysis and design process.

Fig. 7 illustrates the architecture of the SERVUS Use Case Server (UC Server). There are the following basic architectural decisions and characteristics of the SERVUS UC Server:

- The server is realized as an application of a Web-based content and community management framework (here: WebGenesis®) that supports the implementation of HTML5-based collaborative applications by rich generic functions.
- Use cases and all other elements following the meta-model (e.g. requirements, capabilities) are persistently stored in a relational database as WebGenesis® entries. This enables the immediate use of built-in search functions for use cases and the other elements.
- The database structure is organized according to the use case meta-model, which acts in this case as an ontology. Due to the built-in ontology support of WebGenesis® this design decision increases the flexibility of the system and facilitates the linkage of these core elements to thematic ontologies.

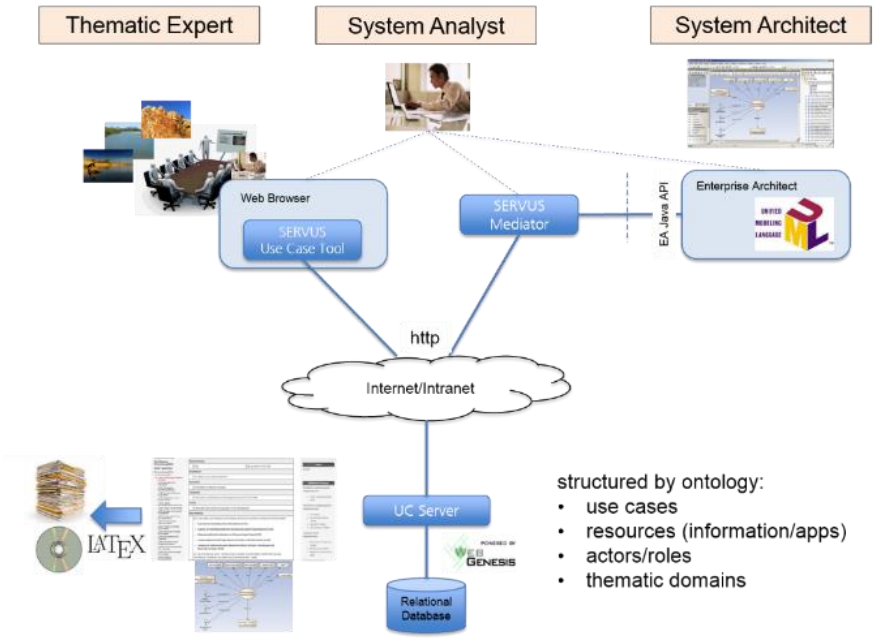


Fig. 7: System Architecture of the SERVUS Use Case Server

- At any time, there is the possibility to generate document reports out of the content of the UC Server. In most projects, there is a need to deliver written use case specifications as soon as a given milestone is reached. By using LaTeX as document generation tool, the administrator may easily change the structure and the layout of the report and adapt it to the requirements of a given project (e.g. addition of project or company logos or compliance with corporate designs).
- The face to the thematic expert as one of the user roles is the Web browser hence, no installation on the user site is necessary. Furthermore, due to WebGenesis® as underlying framework, the layout of the Web-based application may be tailored to the corporate or project design constraints. Furthermore, user management and collaboration utilities between users (e.g. calendar functions) are already available, and may be easily integrated in order to support the analysis process within and across organizational boundaries.

On the other side of the user role spectrum there is the system architect whose objective is to formalize the requirements as UML use case diagrams (e.g. using a UML tool such as the Enterprise Architect) and map the use case scenarios to call sequences of interfaces and services, possibly also specified in UML.

The system analyst has to mediate between both communities: On the one hand, he/she shall facilitate the requirements analysis process within the community of the thematic experts aiming at getting consolidated semi-structured use case models. On the other hand, he/she shall guarantee the consistency with the UML models required for the system design activities led by the system architect. For this purpose the SERVUS UC Server provides a mediator tool that, among others, extracts UML use case diagrams from the Enterprise Architect and feeds them as associated figures into the use case models.

References

Cockburn, A. (2001). Writing Effective Use Cases. ISBN-13: 9780201702255. Addison-Wesley; 2001.

DIN SPEC 91345 (2016). Reference Architecture Model Industrie 4.0 (RAMI4.0). DIN; 2016.

Industrial Internet Consortium (IIC) (2016): *The Industrial Internet Reference Architecture Technical Paper*. Retrieved in April 2016 from http://www.iiconsortium.org/IIRA.htm.

Jacobson, I. and Ng, P.-W. (2005). Aspect-Oriented Software Development with Use Cases. *The Addison-Wesley Object Technology Series*, ISBN 0-321-26888-1; 2005.

Pohl, K. and Sikora, E. (2007). The Co-Development of System Requirements and Functional Architecture. In: Krogstie et al (eds.), (2007): *Conceptual Modelling in Information Systems Engineering*, pp. 229-246, 2007.

Usländer, T. (2010). Service-oriented Design of Environmental Information Systems. PhD thesis of the Karlsruhe Institute of Technology (KIT), *KIT Scientific Publishing*. ISBN 978-3-86644-499-7, 2010.

Usländer, T. and Batz, T. (2011): How to Analyse User Requirements for Service-Oriented Environmental Information Systems. In: Proceedings of the ISESS 2011, Brno, Czech Republic, pp. 161-168, Springer Verlag; 2011.

Usländer, T. (2016). Agile Service-oriented Analysis and Design of Industrial Internet Applications. 49th CIRP Conference on Manufacturing Systems (CIRP-CMS 2016). Published by Elsevier B.V.

Usländer, T. and Epple, U. (2015). Reference model of Industrie 4.0 service architectures: Basic concepts and approach. *Automatisierungstechnik : AT 63 (2015)*, No.10, pp.858-866 ISSN: 0178-2312; 2015.

Wikipedia (2016). https://en.wikipedia.org/wiki/Agile_modeling