FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Model-Driven Engineering for Smart Grid Automation

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Filip Pröstl Andrén, MSc
Registration Number 1229208

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dr. Wolfgang Kastner
Second Advisor: Priv.-Doz. Dr. Thomas Strasser

The dissertation has been reviewed by:

Lars Nordström

Wilfried Elmenreich

Vienna, 24th January, 2018

Filip Pröstl Andrén

Erklärung zur Verfassung der Arbeit

Filip Pröstl Andrén, MSc
Oskar-Jascha-Gasse 61
1130 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24 Jänner, 2018

Filip Pröstl Andrén

For Susanne, Valentina, and Linnea.

Acknowledgements

This thesis is the final result of a work long in progress, and although it is the main outcome, it is still only a small part of the whole effort. Also, as everybody knows who has been in the same situation, a PhD is certainly not a one man job. Since the beginning, many different people have contributed in one way or another. This is for them.

First of all I would like to thank my colleague Thomas Strasser. Thank you for the, sometimes long, and very helpful discussions and your invaluable advice. Without your help and support this thesis would not have happened.

Secondly, I want to thank my supervisor Wolfgang Kastner. I really enjoyed your uncomplicated style and I also want to thank you for your excellent supervision and always very constructive feedback.

Thank you to Matthias, who helped me a lot during the beginning of the work with discussions and ideas. Another thank you is directed to Claudia. The many discussions we had about your PhD also helped me to better understand my own work. Thank you also to Christian, who always provides excellent help when the laboratory is needed.

Thank you Armin, Oliver, Christian, and all the other involved members of the OpenNES project. Much of this work was done during this project and your help, comments, and feedback has made this thesis so much better. Also a great thank you to Jürgen, Christian, and the partners of the MESSE project. It was during the proposal phase of this project that I wrote the main part of the methodology chapter in this thesis. Without the discussions we had for the MESSE proposal, this thesis wouldn't have been the same. Also, to anybody I have forgotten to mention: thank you.

My final thanks goes to my family. To my mother, father, and sister for always supporting me. My deepest thanks to my wife Susanne. For always staying positive and never letting me doubt myself. Finally, thank you to my children Valentina and Linnea for providing just the right distraction when work is hard.

Kurzfassung

Die Einführung von Smart Grid Lösungen hat begonnen, und neue Verfahren kommen in den heutigen Energiesystemen jetzt schon zur Anwendung. Einer der Katalysatoren dieser Entwicklung ist der massive Ausbau von verteilten Energieerzeugern und erneuerbaren Energiequellen in den letzten Jahren. Dies hat zu einem fundamentalen Paradigmenwechsel in Bezug auf die Planung und den Betrieb der elektrischen Energiesysteme geführt. Automatisierungs- und Regelungssysteme, die auf fortschrittlichen Informations- und Kommunikationstechnologien beruhen, sind Schlüsseltechnologien im Umgang mit den neuen Herausforderungen. Das elektrische Energiesystem entwickelt sich von einem Einzelsystem zu einem System von Systemen. Die Umsetzung solch komplexer Systeme von Systemen ist mit einer deutlich erhöhten technischen Komplexität verbunden, die sich auch in erhöhten Lebenszykluskosten niederschlägt. Um diese Komplexität einzuschränken, sind geeignete Automatisierungsmethoden und Werkzeuge für den gesamten Entwicklungsprozess notwendig. Solch fortgeschrittene Methoden fehlen jedoch derzeit.

Diese Arbeit adressiert diese Schwächen durch die Entwurf einer rapiden Engineeringmethode, die alle Phasen des Engineeringprozesses von Smart Grid Anwendungen abdeckt – vom Entwurf der Spezifikation und Design über die Validierung bis hin zur Ausrollung. Das Hauptziel der Methode ist die Verbesserung des traditionellen Engineeringprozesses und damit auch eine Verringerung der notwendigen manuellen Arbeit und der Komplexität. Dafür werden Automatisierungstechniken der modellgetriebenen Softwareentwicklung verwendet. Darauf basierend können vier Entwicklungsphasen identifiziert werden: Spezifikation und Design, Implementierung, Validierung, und Ausrollung.

Das Hauptergebnis ist ein formaler Ansatz für die Spezifikation- und Designphase, zusammen mit einem Konzept für automatisierte Erzeugung und Ausrollung von Anwendungscode und Konfigurationen. Mit diesem Ansatz wird der gesamte Engineeringprozess von Smart Grid Applikationen erheblich verbessert. Eine prototypische Implementierung der rapiden Engineeringmethode wird für eine ausgewählte Smart Grid Applikation angewendet, und schließlich auch in einer Laborumgebung validiert. Die Hauptvorteile dieses innovativen Ansatzes werden durch einen Vergleich von der Leistung dieser Methode mit der Leistung traditioneller Engineeringmethoden betont. Der Vergleich zeigt, dass die entwickelte Methodik dieser Arbeit eine drastische Reduktion der Entwicklungs- und Validierungskomplexität ermöglicht. Gleichzeitig werden der manuelle Aufwand verringert und die Schnelligkeit existierende Engineeringmethoden erheblich erhöht.

Abstract

The rollout of smart grid solutions has already started with new and intelligent methods being deployed to today's power systems. One of the main catalysts for this is the massive deployment of distributed generators from renewable sources in the recent years. This has led to a fundamental change in terms of planning and operation of the electric power system. Automation and control systems, using advanced information and communication technologies, are key elements to handle these new challenges. The electric energy system is moving from a single system to a system of systems. As a consequence, the implementation and deployment of these complex systems of systems are also associated with increasing engineering complexity, which in the end also results in increased total life-cycle costs. To mitigate this complexity, proper automation methods and corresponding tools are also needed for the overall engineering process. Until now, such a method has been missing.

This work addresses these shortcomings with the development of a concept for a rapid engineering methodology, covering the overall engineering process for smart grid applications—from use case design to validation and deployment. The main goal with the methodology is to improve the traditional smart grid engineering process in such a way that manual work and also the engineering complexity are reduced. In order to achieve this automation, techniques from model-driven engineering is used. Based on the model-driven development approach, the methodology consists of four main phases: specification and use case design, implementation, validation, and deployment.

The main result of the work is a formal approach for the specification and use case design phase together with a concept for automatic generation and deployment of target code and configurations for the other phases. Using this approach, the overall engineering process for smart grid applications is greatly improved. The developed rapid engineering methodology is also provided as a prototypical implementation, which is applied to a selected smart grid application and finally validated in a laboratory environment.

This validation also reveals the main benefits of this innovative approach. To show this, the results of the validation are compared to the performance of traditional smart grid engineering methods. The comparison shows that the rapid engineering methodology from this work drastically reduces the engineering and validation complexity for the engineer. At the same time, the manual effort is reduced and the rapidness of current engineering methods is significantly increased.

Contents

CHAPTER 1

Introduction

Smart grids are no longer things only existing in a distant future. New and intelligent approaches for measurement, control, and automation are being implemented in the power systems, even today. This is a trend mostly propelled by the increasing installation of distributed components and energy resources. However, the power systems as known until today were not built to handle this amount of distributed resources. In fact, since these new components cannot be handled centrally, a number of challenging problems are introduced.

However, with the availability of new, intelligent, and robust methods from computer science even the power systems are experiencing a digitalization. In fact, these new methods are seen as key elements to handle the new situation. But, with new approaches also follows new challenges. The future power systems—induced with more and more software applications—require other engineering approaches than the traditional ones.

This work covers one possibility of these new engineering approaches. Especially, it studies where it is possible to support power system engineers using automated procedures. If this is achieved, it is expected to decrease the manual effort associated with the engineering of power system applications today. In the end, the total life-cycle cost of power systems may also be decreased since human effort equals time, and time equals money.

1.1 A Changing Electric Energy System

The electric energy system is experiencing a fundamental change. The traditional power system with central bulk generation providing energy to distributed consumers is not a picture of the reality anymore. Until today, the worldwide power generation was dominated by fossil fuels. The results are increasing CO_2 emissions and global warming, as indicated by the “World Energy Outlook 2013” from the International Energy Agency [173]. In order to mitigate this process and at the same time satisfy a continuously growing demand for electricity, there is an obvious trend towards a sustainable electric energy system [145].

The two main foundations of sustainable energy are renewable energy and energy efficiency [115]. In more and more countries, fossil fuels are being replaced by renewable sources, such as Photovoltaic (PV) and wind generators, biomass, and combined heat-and-power systems [89, 21]. Due to the nature of renewable sources their generation output often follows a stochastic and dynamic model instead of the linear models of traditional power plants. Typical examples are PV or wind, which are directly dependent on the solar irradiation and the force of the wind. As a consequence, in order to estimate the available generation of these sources predicable weather forecasts are needed [91]. Another thing that differentiates renewable sources compared to traditional generation technologies is their size—both in physical size as well as in generation capability. As a result, they are typically available in a much more decentralized way as so called Distributed Energy Resources (DERs) [89].

A higher penetration of DERs is causing other challenges as well, such as reversed power flows. On a nice summer day it is not uncommon that PV plants produce more power than is consumed locally. This leads to negative power flows, which may cause problems since today's power systems are not intended to handle such cases [91]. Secondly, consumers are evolving into so-called prosumers—local energy consumers and producers. They are no longer only consuming energy but also producing, which requires new planning and market solutions [145].

The second foundation of sustainable energy is energy efficiency. In order to increase the efficiency, one main theme has emerged: better coordination of consumption and generation. This has resulted in a number of different solutions, especially for the consumer side, such as demand response, energy storage systems, and smart metering [107, 55]. They all have the goal to only consume energy when it is available [145]. At the same time, the electrification of the transportation sector is also causing new challenges. With Electric Vehicles (EVs), consumers are no longer fixed to a certain graphical location [32].

Accompanying these new technology developments, further research and regulatory alterations are needed. The decentralized manner and also geographical dispersion of renewable sources and customer side solutions introduce challenges, which cannot be tackled only using pure technical solutions. Changes in regulations and grid codes are also necessary [11]. Of course grid codes have always changed, but in recent years the rapid changes in the electric energy system has also affected regulations and grid codes. One example is the so called "50.2 Hz problem" in Germany [50]. Due to this, grid codes were changed and even required already installed inverters to be retrofitted [146].

Summarizing, the above sketched developments are leading to new challenges. Today, power utilities and system operators are increasingly confronted with a highly dynamic and less predictable demand-supply balance. Consequently, the planning, management, and operation of the future power systems have to be redefined.

A sustainable energy system does not only create challenges, it also provides a number of opportunities. A large scale integration of DERs introduces another level of controllability to the power system. Consequently, the management and operation of the future power system can take advantage of new and advanced control functions. An overview of some of the most important functions and services is given below [40, 34, 145]:

- *Advanced monitoring and diagnostics:* Monitoring, state estimation, and self-diagnostic capabilities in the Medium Voltage (MV), and especially in the Low Voltage (LV) distribution grids will be crucial for future power systems.
- *Optimization/self-optimization capabilities:* Fluctuating generation from renewable sources will require self-optimization of future distribution grids. This includes automatic reconfiguration of grid topology as well as self-healing in case of faults in the power system.
- *Adaptive protection:* Automatic or semi-automatic adaption of protection devices in respect to the actual power grid conditions. One example is the adaptation of the protection system settings due to the bidirectional power flow caused by DERs.
- *Distributed automation and control:* Distributed automation and control with modern solutions (e.g. automatic decision finding and proactive fault prevention) have to be provided for power system operators.
- *DERs with ancillary services:* Usage of ancillary services provided by DERs (e.g., local voltage or frequency control) in order to improve power system stability, and at the same time increase the hosting capacity in power distribution grids.
- *Demand response/energy management support:* The use of electric loads and energy storage systems is increasing, which provides further operation flexibility.

Additionally, changing framework conditions—such as the liberalization of the energy markets, new regulatory rules, and technology developments—also require new adaptations to the planning, management, and operation of the future power system. However, in order to implement the functionalities listed above sophisticated component design methods, intelligent Information and Communication Technology (ICT) architectures, automation concepts, as well as proper standards are necessary. They are seen as key elements in order to manage the higher complexity of the future intelligent power systems, also know as smart grids [34, 54, 158].

It can also be expected that the complexity of the smart grid will not stabilize, but increase further. Initial research towards interconnection with other complex systems is already under way. Prominent examples are hybrid grid approaches, where the electric energy system is coupled with other energy grids, such as gas or water [165]. Another example is smart cities, where smart grids are only one component of a large optimal and sustainable system [20, 32]. As a consequence, the electric energy system is moving towards a complex cyber-physical system of systems [145].

1.2 Problem Formulation and Research Question

The development of these new smart grid systems are associated with increasing engineering complexity, resulting also in increased total life-cycle costs. The traditional engineering methods used for power system automation were not intended to be used for applications of this scale and complexity. When studying different smart grid projects and activities the engineering methodology can be grouped into four phases: design, implementation, validation, and deployment [41, 19, 118, 153]. Some may argue that a system must be deployed before it can be validated. However, for power systems this is almost never the case. The electric energy systems are categorized as critical systems, where failure or malfunction may lead to personal damage or great economical cost. Consequently, new components or functions must be extensively validated and tested before deployed and commissioned [153]. The phases of the traditional engineering method for smart grids are illustrated in Figure 1.1.

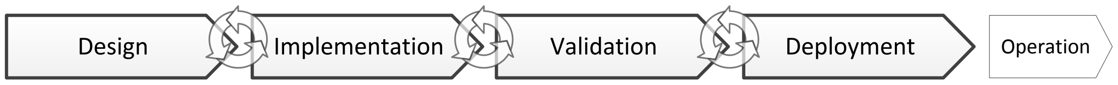


Figure 1.1: Traditional engineering process for smart grids.

Related to this process a number of open issues regarding smart grid engineering can be identified [175, 5, 8]:

- *Rigorous engineering:* Rigorous model-based engineering concepts for smart grid applications are missing or only partly available.
- *Increased effort:* More and more complex applications and solutions result in increasing engineering efforts and costs. Methods are needed to increase the effort of traditional smart grid engineering.
- *Multidisciplinarity:* The multidisciplinary character of smart grid applications requires engineers to have an expert knowledge in each discipline. When this is not the case, it increases the risk of human errors.
- *Flexibility:* From today's point of view it is nearly impossible to address all future needs and requirements in smart grid systems and DER components. The flexible addition of new functionalities in grid components is therefore an essential feature.
- *Proprietary legacy systems:* System operators expect a long service life of all components in their systems. Available proprietary automation solutions in smart grid systems prevent efficient reuse of control software.
- *Geographical dispersion:* The distribution of components over large geographical areas requires special attention. New ICT approaches and wide-area communication are needed.

- *Timing and performance constraints*: Some applications may enforce real-time constraints on hardware, software and networking. Performance management is often not adequately addressed in existing engineering processes.

- *Scalability:* Current engineering methods are focusing on the development for a single system. With a more complex smart grid, these methods must not only be able to handle a single system, but a system of systems.

To tackle these issues, new engineering methods are needed. With the usage of proper methods, automation architectures, and corresponding tools, there is a huge optimization potential for the traditional engineering process of smart grid applications.

One method to increase engineering efficiency is to start with detailed use case and requirements engineering. This has led to a number of recent smart grid projects where use case descriptions of corresponding applications are in the focus [127, 10]. Different methodologies and standards exist today for describing use cases. Two of the most common methods in the smart grid domain are the Smart Grid Architecture Model (SGAM) [25] and the IEC 62559 approach [70] (formerly known as the IntelliGrid method). The main aim of use case descriptions and corresponding derived requirements is to provide a clear and concise documentation of the application [9].

The result of this methodology is a structured description of use cases, often containing a very high amount of information. However, since this information is still in a non-formal representation, it cannot be adequately utilized in a computerized and automated approach. The same is true for other specifications that are typically provided as an input to the engineering process. Also, there is no standardized way of representing the objects (e.g., controllers and power grid components), neither in the way they are depicted, nor in the semantics used in the description.

Consequently, a significant amount of work has to be spent a repeated number of times: *(i)* in the design, *(ii)* the implementation, and also in *(iii)* the validation phase. This is a very time-consuming and error-prone approach to engineering smart grid applications. However, if the information gathered during the use case description phase can also be used directly in an automated way, the development effort can substantially be decreased. By collecting the use case information in a formal model this can be used for direct automatic code generation. The result can be executable code for field devices, communication configurations as well as documentation. Moreover, an automated approach also has the potential to decrease implementation errors and at the same time to increase the software quality [96, 132].

However, up to now, there is no integrated engineering method available, covering the whole development process for smart grid applications, which fulfills this goal. Exactly this gap is what this work is trying to fill. Therefore, based on these observations, the main research question of this thesis is formulated.

Research Question:

Given the traditional engineering process for smart grid applications—covering use case design, implementation, validation, and deployment—what can be done to significantly reduce the amount of manual work needed from the smart grid engineer(s)?

As stated by the research question the proposed rapid engineering methodology should be applicable for the whole development chain of smart grid applications. Specifically meant are the four main development phases: design, implementation, validation, and deployment. These phases are general and can always be found independently on the used engineering methodology, agile or plan-driven. A high-level hypothesis can be formulated that tackles the research question by focusing on these phases and the transitions between: *Automation during the engineering phases, and of the transitions between the different phases, reduces the amount of manual engineering work needed for smart grid applications.*

This hypothesis is too vague to be evaluated within this thesis. Instead, a more specific hypothesis must be stated. First of all, automation can be done in many different ways. Here, Model-Driven Engineering (MDE) [121] will be used for this purpose. In Section 2.5.1, MDE is explained in more detail. For now, it suffices to say that it is a software engineering methodology, which focuses on the development of models and the automatic transformation between these models. These models are virtual representations, either of real-world objects or of software artifacts. Seen from a more generic perspective, each engineering phase can be represented with one or more models. Using this knowledge a more specific hypothesis is formulated.

Research Hypothesis:

Model-Driven Engineering of smart grid applications, with model transformations during the engineering phases—and for the transitions between the different phases—will reduce the amount of manual work needed to describe information in multiple models.

It is the intention that the methodology in this thesis will reduce the amount of manual work that is needed to describe information in multiple models. In other words, instead of engineers describing the same information (e.g., functionality) in multiple phases of the engineering process, this should be done automatically through model transformations (e.g., code generation). To show that the hypothesis holds, this thesis presents the development of a rapid engineering methodology for smart grid applications. The methodology is based on techniques from MDE and covers the engineering phases design, implementation, validation, and deployment.

1.3 Scientific Research Method

The research method in this thesis follows the inductive-hypothetical research method [136], which is a typical top-down approach for this kind of studies. According to Omona et al. [106], the inductive-hypothetical research strategy combines theory and practice and emphasizes problem specification from a multidisciplinary point of view. The method consists of five steps [136]:

1. *Initiation*: Based on the current situation, observations of the state of the art is carried out. The result is a descriptive empirical model. In this case, this means that related work on smart grid engineering methods is studied.
2. *Abstraction*: Based on the empirical model, use case studies are carried out to create a descriptive conceptual model. Here, this corresponds to requirement identification based on a number of business cases.
3. *Theory Formulation*: Following the conceptual model, a theory is formulated, resulting in a prescriptive conceptual model. In this thesis, this model is the rapid engineering methodology.
4. *Implementation*: In this step, the theoretical model is implemented and applied on the example cases, which results in a prescriptive empirical model. For this thesis, this means that a prototypical implementation of the rapid engineering methodology is created and also that this is applied to a test case.
5. *Validation*: The last step is a validation of the implemented solution and comparison with the state of the art before the study. In this case, the results from the application of the rapid engineering methodology are compared with other methods.

The last step, which is the validation and evaluation of the research in this thesis, should be discussed in more detail. Since the engineering method for smart grid applications is a practical process, it is not suitable to validate it using traditional evaluation methods for physics, biology or medicine [130]. Instead, validation methods from software engineering are better suitable. Shaw presents a number of different validation approaches for research in software engineering [131]. These were categorized based on research question, research result, and research validation and are summarized in Figure 1.2.

In this thesis, the research approach follows the path: *development method*, *procedure, technique*, and *evaluation*. Questions in the *development method* group cover research questions like “What is a better way to do/create X?” [131]. This fits very well with the main research question in this thesis, as seen in Section 1.2. Based on the research question, a research result will be reached. Results in the group *procedure, technique* are for example “New or better way to do some task, such as design, implementation, ...” [131]. This can be compared to the specific hypothesis in this work. The final step is the validation, where the choice should be made with regard to the type of

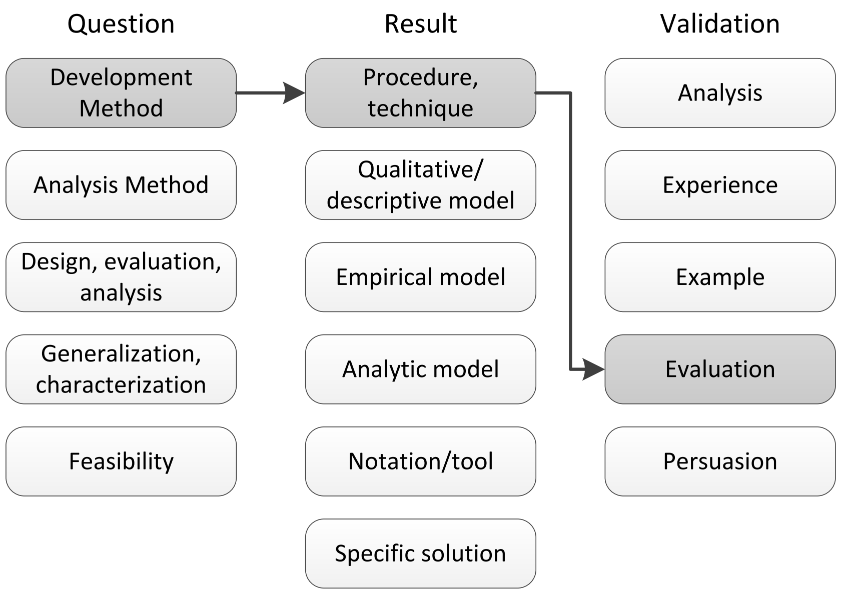


Figure 1.2: Research validation methods for software engineering, summary from [131]; highlighted options show the plan for this thesis.

research question and result. Here, *evaluation* means validation by comparing the rapid engineering method with other traditional smart grid engineering methods. Based on the application of the rapid engineering method on a test case, the amount of manual work needed from smart grid engineers is evaluated (see Research Hypothesis). This will be compared to the amount of manual work needed for the traditional engineering methods.

1.4 Structure of the Thesis

Using the scientific research strategy presented above it can also be as a framework for the structure of this thesis. Therefore, the following chapters in this work are directly related to the five main steps of the research method. Below is a short summary of each of these chapters. Furthermore, for some chapters contributions have already been published in journals or at conferences. These publications are also mentioned below.

Chapter 2 presents the current state of the art on the topic of engineering support for smart grid applications. Related work is presented in different sections, each associated with one of the four main engineering phases (i.e., design, implementation, validation, and deployment). Furthermore, this chapter discusses both work that is considered a prerequisite for understanding this work, and also work that is considered related or competitive to this thesis.

Chapter 3 abstracts the state of the art using a number of use cases, or in this case business cases. These are used to exemplify different problems typical to engineering of smart grid applications today, and also to suggest possible solutions. The main outcome of this chapter is a number of requirements that are used as basis for the conceptualization of the rapid engineering methodology. Especially, parts of these requirements were already published in [117].

Chapter 4 is the main chapter of thesis. It contains the formulation of the rapid engineering methodology. After a general concept is presented, appropriate modeling and design methods are discussed. Thereafter, the work is divided into sections describing the methodology for each engineering phase (i.e., design, implementation, validation, and deployment). The main work of this chapter is based on [5] and [117], but also contains aspects from other publications, especially [3, 9, 6, 8], and [116].

Chapter 5 shows how a prototypical implementation of the rapid engineering methodology can be created. For each of the different engineering phases, it is described what tools were used and what adaptations were implemented. This chapter is partly based on [117], although much more details are provided in this work.

Chapter 6 uses the prototypical implementation to apply the rapid engineering methodology to a specific test case. In this chapter, a detailed description is given about how the rapid engineering method is intended to be used. Furthermore, this chapter also shows how different type of validation methods can be integrated and used. The main parts of this chapter were already published in [117].

Chapter 7 is the last chapter of the thesis. It contains two parts. The most significant part is an evaluation of the rapid engineering methodology and a revisit to the main research question as well as the Research Hypothesis of this work. Specifically, the rapid engineering method is compared to two other traditional smart grid engineering approaches. A comparison is made with respect the amount of manual effort needed for each approach. This evaluation is also partly found in [117]. The second part of this chapter are the conclusions, where the main contributions of the thesis are summarized and possible directions for future research addressed.

CHAPTER 2

State of the Art

The main goal of the work in this thesis is to find a methodology that supports the engineer with the design, implementation, validation, and deployment of smart grid applications. Engineering support is nothing new or unique for the smart grid domain. In other domains, this topic is even more advanced than for smart grids. With this in mind, this work uses an approach where well-proven engineering technologies from other domains are reused and applied to the smart grid engineering domain.

This chapter provides a discussion on the current state of the art for the topic of this thesis. The work is selected according to topics associated with the four main engineering phases: design, implementation, validation, and deployment. For each phase, both the current practice is presented and an indication of the future is provided. As already indicated by the Research Hypothesis, MDE technologies are important for the realization of the rapid engineering methodology. Consequently, this chapter also provides an overview of the MDE concept, together with other software engineering methods used in this work.

Finally, the chapter is concluded with a summary of other MDE approaches for smart grid application development. These are concepts where MDE has been used in one way or another to improve the engineering effort of smart grid applications. For each concept, it is shown how this work differentiates and goes beyond the current state of the art.

2.1 System Engineering Approaches

The main intention with system engineering approaches is to help and guide the engineers and project managers during the development of a system. By structuring the development approach, it is possible provide distinct methods and solutions for specific tasks. Such engineering approaches have always existed, but since the introduction of software development the number of different methods have increased [37].

Generally, engineering approaches can be divided into two categories: plan-driven and agile methods. Traditional engineering methods, typically used for hardware or manufacturing purposes, often belong to the plan-driven methods. In comparison, modern software engineering approaches usually fall into the agile category [15].

The following sections summarize some existing engineering methods. The main idea is to give the reader an overview of common engineering approaches. Furthermore, the presented engineering approaches are also compared with the engineering approach chosen for this work in order to show how they are related and what the main differences are.

2.1.1 Plan-Driven Methods

Plan-driven methods are often coupled with an extensive requirement engineering phase before anything has been implemented. This is typically the case for large and complex projects, where plan-driven methods provide for better communication and coordination across large groups. For such cases, the prior planning, design and specification are often required to meet certification standards [15]. There are multiple approaches following this approach. In this section, two of the most well known approaches are summarized.

The waterfall method is probably the most well known process for system engineering. One of the reasons for this is that it is easy to understand. Another is that it intuitively follows the most obvious way to develop a system. The classical waterfall model defines a non-iterative engineering process divided into a number of phases. Consequently, this does not allow any room for errors during the development phases. Thus, event though the waterfall method is easy to understand it does not follow a practical model for actual development projects [93]. Nevertheless, the development phases used by the waterfall method are general and embedded into many other engineering methods—plan-driven as well as agile. Therefore, in order to understand what an engineering approach is, it is important to know the waterfall method. Usually, the following phases are used [93]:

1. *Feasibility*: The main aim of this phase is to analyze if it is feasible, technically as well as financially, to develop the intended system.

2. *Requirements*: During this phase requirements are analyzed and specified. The main aim of the requirement analysis is to collect all relevant requirements from the customer and document them. Thereafter, these requirements are further specified, usually divided into functional and non-functional requirements.

3. *Design*: The main aim of this phase is to transform the requirements into a structure that is suitable for implementation. For a software product, this could for example be a software architecture.

4. *Implementation*: After the design phase, the design is implemented. For the above mentioned example of a software product, this would to translate the software design into source code.

5. *Testing*: The aim of the testing phase is to evaluate and validate the implementation, for example, according to a system test plan.

6. *Maintenance*: The last phase considers the installation, migration, support, and maintenance of the complete system. Especially for software products this phase usually requires the most effort.

The main shortcoming with the waterfall method is that it is too idealistic. As already discussed, it assumes that no development error is ever committed during any of the phases. This is clearly never the case—and would also make the *testing* phase obsolete.

One approach to overcome the shortcomings of the waterfall model was introduced by the V-model. Instead of only using a top-down approach with no iterations it follows a "V", where the first four phases of the waterfall model are located on the "\" and the testing and maintenance are located on the "/", as seen in Figure 2.1. The project testing and integration receives considerable more attention than in the waterfall model. Furthermore, iterations are possible—or even intended—since the tests are validated against the corresponding part in the project definition phase [46].

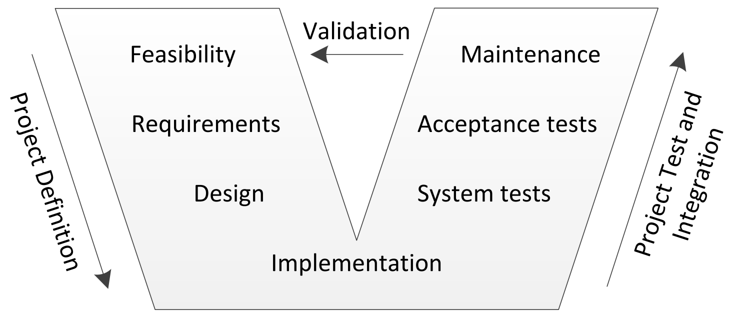


Figure 2.1: System engineering process according to the V-model.

While the V-model benefits from the same simplicity as the waterfall model it also has some drawbacks. Two of the most common points of criticism, not only for the V-model but in general for plan-driven methods, are the rigid structure and the inability to react to changes of requirements [14]. As a response to this, agile methods were introduced.

2.1.2 Agile Methods

Agile development is an umbrella term for a set of methods and principles focusing on development where requirements and solutions evolve during the project time. Strong collaboration between development groups and customers is motivated. The main basis for agile development was started by "The Agile Manifesto" in 2001 [48]. It was created as a common denominator for the new software development methods that were receiving a growing interest. As with the plan-driven methods, multiple established agile methods exist. Below is a summary of two of the most commonly used methods today.

Scrum is probably the most well known agile method. It is based on a set of core values, principles, and practices—collectively the Scrum framework. The main idea with Scrum is not to be a standardized process, but more a framework for organizing and management work. This framework is made up of specific roles, activities, artifacts, and rules [125].

Three main roles make up a Scrum team: product owner, Scrum master, and development team. The product owner represents the product's stakeholders and is responsible for what will be developed and in which order. The Scrum master is responsible for guiding and leading the team in following their approach within the Scrum framework. The responsibility of the development team is to decide how to implement and deliver what the product owner has asked for [125].

The main Scrum activity is the sprint. It defines a time, usually up to a month, during which the development team implements a number of features. In order to decide which features should be implemented, two backlogs are used: the product backlog managed by the product owner and the sprint backlog. The product backlog contains all items that the product owner wants to have in the final product. The number of items usually exceeds the number of features that can be implemented during one sprint. Therefore, before a new sprint begins a subset is selected by the development team for the sprint backlog. Thus, the sprint backlog consists of a number of chosen features that should be implemented during this sprint. The goal of each sprint is to have a functional prototype containing the newly implemented features. After the sprint, this is inspected together with the stakeholders in a sprint review. Then the cycle repeats, beginning anew with the development selecting a new subset of features for the sprint backlog [125].

Another agile approach is the Kanban Method. Compared to Scrum it does not define principles for how a team should produce a product. Instead, it is more a method for visualizing the flow of work, where the main goal is to balance demand with available capacity and to spot bottlenecks. To do this, a kanban system is used. It is a flow system that limits the amount of work in progress by using visual signals, usually represented on a kanban board [2], as seen in Figure 2.2.

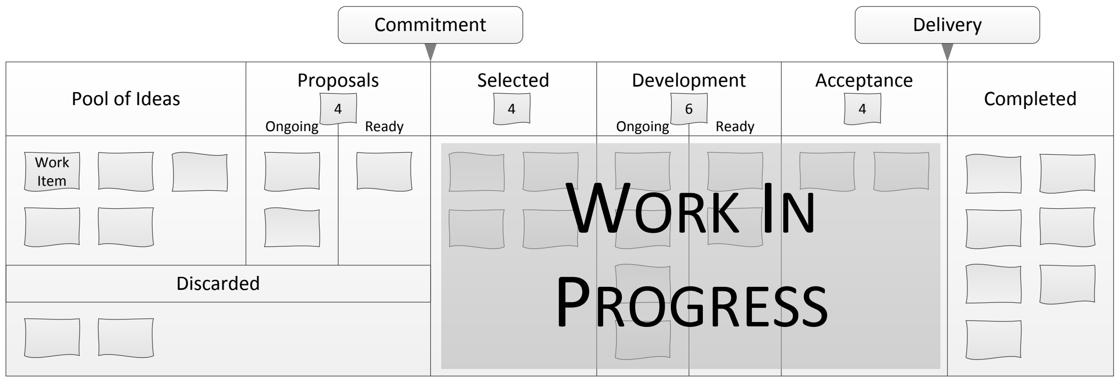


Figure 2.2: Example of a kanban board [2].

The visual signals on the kanban board are used to handle the work in progress. They prevent too much or too little work from being implemented at the same time. Ultimately, this should improve the flow of value to the customers. As described by Anderson et al.: “Work is “pulled” into the system when other work is completed ... rather than “pushed” into it when new work is demanded” [2]. Work items are generally moved from the left to the right as they are implemented. At the head of the board, cards are used to display the limit for each columns. This also defines the limit for the work in progress. In addition, the board also shows which points are identified for commitment and delivery.

2.1.3 Importance for Rapid Engineering of Smart Grids

Smart grid engineering requires methods that can handle both hardware and software development. Traditionally, it was a field where hardware dominated, but where software is gaining more and more ground today. Thus, the agile engineering methods also become more and more interesting for smart grid engineering [41].

In Section 1.2, the main research question of this thesis stipulates four engineering phases—design, implementation, validation, and deployment—which will be studied during this work. Principally, these match with the different phases of the waterfall and the V-model. However, they can also be found in the agile methods. For each feature that is implemented during the sprint of the Scrum method, all these phases are carried out. The same applies for the Kanban model. Once a work item has been selected it will also go through these phases. Consequently, the findings of this work are not restricted to only one type of engineering method, but can be used independently of the chosen development model.

2.2 Use Case Design and System Description

Before an application can be implemented it needs to be clearly designed and specified. This is independent of the followed engineering method. For plan-driven methods, a design and specification is done for the whole application, whereas for agile methods smaller parts (i.e., work items or features) are considered. Often accompanying the design is also a system or architecture description. For smart grid applications, this is typically a model of the power system and the corresponding ICT and automation infrastructure. This section summarizes a number of use case design, and architecture description methods that are commonly used for smart grid applications. Furthermore, a perspective from other domains is also given.

2.2.1 IntelliGrid Use Case Template—IEC 62559

The IntelliGrid methodology was originally developed by Electric Power Research Institute (EPRI) in 2003 as a response to the increasing complexity of power system automation [62]. The idea was to use a systems engineering approach for defining the future energy systems applications. Since then, the IntelliGrid method has become its own standard described in the IEC 62559 [69], which is one of the most commonly used methods for describing smart grid use cases [122]. Using existing approaches from systems engineering, it integrates requirements engineering and best practices and combines them into a process model. Furthermore, the methodology explicitly addresses the identification of stakeholders and how to structure communications in a project [122]. The core of IEC 62559 identifies five engineering phases [69]:

1. Approval and review of business cases
2. Description of user requirements with use cases
3. Identification of detailed requirements based on the previously developed use cases
4. Evaluation of standards, technologies, and best practices for project applicability
5. Development of technical specifications based on the previous results

The first three phases are especially covered by the IEC 62559. In fact, a use case template is provided. The environment of the use case is described by documenting the relation towards other actors, legal issues, as well as preconditions and assumptions that affect the function. The use case itself is described through a narrative as well as a visual representation (e.g., a Unified Modeling Language (UML) diagram). Furthermore, a detailed step-by-step description of the use case is provided. For each step, a triggering event and the resulting information exchange is documented. The information exchange is described by its content, producer, consumer, and any associated requirements [69, 122]. Appendix B contains a number of examples described according to IEC 62559.

Although the IEC 62559 use case template provides a structured approach, which greatly improves the development of smart grid use cases, it still has its drawbacks. One of these is the collaboration between different stakeholders. The original template is intended to be described in text documents making them difficult to compare and harmonize. As a solution, to this Trefke et al. suggested a use case management tool—the Use Case Management Repository (UCMR) [162]. It is a web-based tool that supports the development of smart grid use cases according to the IEC 62559 template. This allows simultaneous collaboration between experts and organizations [51].

The IEC 62559 has become a de facto standard for smart grid use case descriptions. It provides a structured and practical approach, which is not only comprehensive for domain experts. Furthermore, with the help of tool support it is now also possible to effectively collaborate between different stakeholders. Based on this argumentation, the IEC 62559 approach is a method that should be supported by the rapid engineering methodology in this work. However, even though a tool support is available, the current IEC 62559 approaches are not directly suitable for an automated approach. In order to use the information from the use case in an automated approach, the descriptions must be machine-readable. The UCMR certainly introduced more formalism than the standard text documents, but main description of the use case is still a narrative written in prose. Consequently, this information is very complex to process for a computer. Secondly, the current approaches are mainly intended for documentation purposes and thus not directly appropriate for code generation.

2.2.2 Smart Grid Architecture Model

The SGAM was created as a result of the "European Commission M/490 Standardization Mandate to European Standardization Organizations" [34]. Initially it was mainly intended for the coordination of standardization activities. The idea was to use the SGAM model to identify gaps within and amongst standards, but also topics requiring cooperation among standardization organizations [34]. But, the SGAM also has the potential to provide a structured approach for modeling of smart grid use cases [25].

The basis for SGAM is a three-dimensional framework consisting of domains, zones, and layers. In the domains, the traditional layout of the electrical energy infrastructure can be found: generation, transmission, distribution, DER, and customer premises. The zones depict a typical hierarchical power system management: market, enterprise, operation, station, field, and process. These two axes form the component layer. On top of the component layer, four interoperability layers are placed: the communication, information, function, and business layers [25]. Combined together, these components create a three-dimensional model, as seen in Figure 2.3.

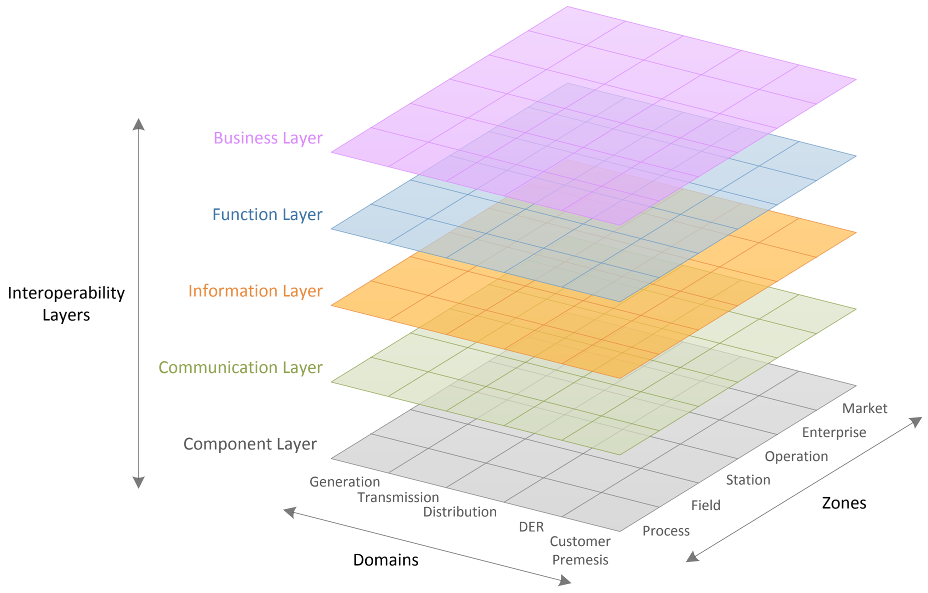


Figure 2.3: Representation of the SGAM, based on [34].

Accompanying the framework in Figure 2.3 is also a use case design methodology. It is related with the IEC 62559 use case template, which is used as a basis for describing use cases. Once a use case has been described using IEC 62559 it is subsequently mapped into the different layers of SGAM. In order to do this in a structured way, the following design steps are defined [24, 26]:

1. *Use Case Analysis*: The first step is an analysis of the use case. It is suggested to use the IEC 62559 template to create an initial use case description [26].
2. *Business Layer Design*: The business processes, services, and organizations, which are linked to the use case, are mapped to the business layer. These business entities are placed in the appropriate domain and zone.
3. *Function Layer Design*: In the function layer, functions and their interrelations should be represented. The functions are derived from the initial use case description. A use case can be hierarchically divided into sub use cases and functions.
4. *Component Layer Design*: After the business layer and the function layer have been modeled, they have to be matched with a certain system. Thus, the next step is to model the component layer. Based on the actors involved in the use case, and any existing system components, the needed components for the use case can be derived and assigned to a domain and zone. Subsequently, the derived functions from the function layer can be assigned to a corresponding hardware.
5. *Information Layer Design*: In the information layer, the information exchanged between functions, services, and components is represented. Information objects can be identified by analyzing the data exchanged between actors involved in the use case (e.g., using sequence and activity diagrams). Another important aspect of this layer is to represent which data models are used for the information exchange.
6. *Communication Layer Design*: Taking the exchanged information and data models identified in the information layer into account, suitable communication protocols and ICT techniques have to be identified. These should be represented in the communication layer.

One of the main advantages with the SGAM approach is its coordinated set of viewpoints. It allows to depict various interrelated aspects of smart grid architectures and supports the identification and coordination of elements on different levels. Using the different viewpoints, it is possible to identify interoperability issues and interrelated aspects are easier discovered [163].

From an engineering point of view, most effort is put into the first step (i.e., the use case analysis). In order to support the user with the methodology described above, Dänekas et al. developed the so-called “SGAM Toolbox”, a UML-based Domain-Specific Language (DSL) available as an extension to the Enterprise Architect software [31]. One advantage of the SGAM Toolbox is that common UML modeling tools like sequence and activity diagrams are available since these are standard parts of Enterprise Architect.

Consequently with the SGAM Toolbox, only one tool is needed covering all steps in the SGAM methodology. The main idea of the SGAM Toolbox is to provide support for traditional use case descriptions. Furthermore, it is also possible to use the code generation capabilities provided by Enterprise Architect [103].

In summary, SGAM is a structured way to approach smart grid architecture development. However, without tool support the real benefits cannot be utilized. The SGAM Toolbox was the first attempt at creating such a tool. From this point of view, it would also be a good candidate for the design phase of the rapid engineering methodology. However, some disadvantages exist. First of all, even if code generation is possible it is not intended for smart grids. Furthermore, there is no special support to handle interconnected distributed applications. For such applications the interaction (e.g., information model, communication protocol) between the components is of importance and often needs to be specifically configured. Neither the SGAM Toolbox nor Enterprise Architect support the automatic generation of such configurations.

2.2.3 The Common Information Model—IEC 61970 and IEC 61968

Since deregulation of the energy market there has been an increasing need for power companies to exchange data on a regular basis. The main reason for this is to ensure a reliable operation of the interconnected power networks that are owned and operated by different utilities. Traditionally, power companies all use different formats to store their data (e.g., topological power system network data, work scheduling information or static simulation software files). This data needs to be exchanged, both internally and externally with other companies. The complexity of exchanging this data using different formats has driven the requirement for a common format that covers all the areas of exchange for smart grid systems. This is one of the main drivers for the Common Information Model (CIM) [95].

As with the IntelliGrid method, CIM also has its roots from EPRI in the mid 1990s. In a first step, CIM was an internal database model for Energy Management System (EMS) and Supervisory Control and Data Acquisition (SCADA) systems. Over time, it evolved into a powerful, object oriented data model also including interfaces for power system applications. The most recognized part of this framework is a large information model being a domain ontology for the energy sector. This model is platform- and technology-independent, maintained in UML. Although, the common goal is to enable semantic interoperability each series addresses a certain application area. Today, it is in the scope of utilities, vendors, and system providers along the whole energy value chain—from generation to consumption [166].

Since then CIM has evolved into its own International Electrotechnical Commission (IEC) standard. In fact, the standard series consists of three different parts: the IEC 61970, the IEC 61968, and the IEC 62325 standards. The focus of IEC 61970 is the power transmission system, in particular its EMS. The main objective of this standard series is to model power grid topologies. Complementary, IEC 61968 covers data for Distribution

Management Systems (DMS) in power distribution grids. Finally, IEC 62325 is the latest extension for defining CIM-based messages for market communications [68, 67, 123].

Complementing the ontology CIM also defines serialization and technology mappings. As already mentioned, the data models are maintained in UML and are thus technology independent. In order to make them applicable it has to be specified how the data models can be modeled with certain technologies. For this purpose, the Resource Description Framework (RDF) was chosen [123].

The CIM approach is widely used for modeling power system topologies. However, in a holistic approach as the rapid engineering methodology it covers only a part, the power system topology. It is not in the scope of CIM to model ICT and automation infrastructure. Neither is it intended to be used to model automation or control functionalities. For this reason, it needs to be combined with other methods.

2.2.4 Use Case and System Description in Other Domains

Two commonly known methods for system descriptions are UML [164] and the System Modeling Language (SysML) [105]. Although both methods are used for system design UML mainly focuses on the description of software systems, whereas SysML is more intended for the description of physical systems. Both of these methods are also used for smart grids, however not as extensively as for example in software engineering.

Combined together, UML and SysML offer a wide portfolio of methods and tools for system and software modeling. Both methods are general-purpose modeling tools, which means that they can be use to model any type of system. However, when only considering smart grids, this is also one of their main disadvantages. It would require of the smart grid engineer to learn another language in order to model the power system. Secondly, it may lead to misunderstanding when a domain specific notation cannot be used.

Especially in software and embedded systems engineering there exist a multitude of Architecture Description Languages (ADLs). There, they are used as a formal language to create a description of a software architecture. Prominent examples are the Architecture Analysis and Design Language (AADL), which was initially intended for real-time and performance-critical applications in avionics [42], and AUTomotive Open System ARchitecture (AUTOSAR), an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers [49].

Although there is much to learn from these ADLs, there still does not exist a corresponding language for smart grid applications. Furthermore, what these ADLs all have in common is that they focus mainly on the software architecture (i.e., the number of processors, buses, memory) and how processes and tasks are distributed for this architecture. There is no real possibility to represent the system components, which in the case of the power system would be, for instance, switches, transformers, loads, and generators.

2.3 Power Utility Automation

This section covers state of the art associated with the implementation of power utility and smart grid automation functionality. From a chronological point of view, this is the phase that follows after the use case identification and design phase. In general, two main topics are covered in this section: programming techniques and ICT approaches for smart grids. In both cases, only an excerpt of the available existing work is presented. Focus is mainly on standards and techniques of high importance for the development of the rapid engineering methodology. Either they will be used directly or they are methods that at first may seem the best choice, but at second glance reveal themselves inappropriate for the task at hand.

2.3.1 Programmable Controllers—IEC 61131

The standard IEC 61131 with the name “Programmable Controllers” was originally published in 1990s. Until then each manufacturer of Power Line Carriers (PLCs) and other controllers all had their own programming approach. Of course, this prevented people from using devices and controllers from different vendors since they were not compatible with each other. In order to solve this problem, and thus prevent a further vendor lock-in, one of the main goals of the IEC 61131 was to unify the PLC architectures and standardize the programming approaches [76]. Today, nearly every PLC vendor supports IEC 61131, at least partially. As a consequence, engineers who are experienced in IEC 61131 will experience less problems when moving from one vendor to another. Nevertheless, vendor-specific extensions are still very common, which makes a direct reuse of control software practically not possible [177].

Until now, nine different parts of IEC 61131 have been published. The most prominent part is the IEC 61131-3 where four different programming languages are defined. These are divided into textual (Instruction List (IL) and Structured Text (ST)) and graphical (Function Block Diagram (FBD) and Ladder Diagram (LD)). On top of these languages, a Sequential Function Chart (SFC) was defined. It has elements to organize programs for sequential or parallel execution. Apart from the languages themselves, the IEC 61131-3 part also defines common elements for all languages. These include declarations of data types and how information is exchanged within a program [63].

Applications programed using one of the languages in IEC 61131-3 are executed using a cyclic model. At the beginning of the cycle, the process inputs (e.g., from sensors and measurement devices) are recorded. Thereafter, these inputs are made available to the program. In the next step, the program is executed. Depending on the type of language, and vendor interpretation, the order of execution can vary. Usually a top-down and left-to-right approach is used [177]. Once the program has been executed, the results are written to the outputs of the controller. Immediately thereafter, the output signals are available to the process. After this step, the cycle starts over from the beginning [63].

At the time when the IEC 61131 standard was developed centralized control approaches were still state of the art of software engineering. Therefore, the standard’s main focus is the programming of single PLCs. In other words, a control application is always dependent of the hardware platform. Consequently, it is not possible to model the application platform-independently. For distributed control systems, this is often a big disadvantage. It forces the engineer to decide about the location (i.e., execution platform) of a function before the implementation has started.

IEC 61131 is not only used in the automation industry, it is also used in many power system automation devices [155, 154]. Since the smart grid is in every sense a distributed system, the usage of IEC 61131 for future power system applications is questionable. With more complex and interconnected applications, higher demands will also be put on the modeling and design methods. For such cyber-physical energy applications, the device-centric view of IEC 61131 is no longer suitable [60, 146].

2.3.2 Distributed Control Reference Model—IEC 61499

An approach to handle the increased complexity of the next generation of automation systems is provided by the IEC 61499 standard [64]. The IEC 61499 reference model has been developed especially as a methodology for modeling open and distributed industrial-process, measurement and control systems. A further goal was to obtain a vendor-independent approach for modeling and programming automation applications. Apart from this, the IEC 61499 standard also focuses on three main issues [86]):

- *Portability:* This is the ability of software tools to correctly accept and interpret library elements produced by other software tools.
- *Configurability:* It is the possibility to configure (control) devices, and their software components (i.e., select, assign locations, interconnect and parametrize), using multiple software tools.
- *Interoperability:* This is the ability of (control) devices from different vendors to interact with each other, thus executing the functions specified by one or more distributed applications.

The use of IEC 61499 to implement automation functions for smart grids has already been suggested in a number of publications [60, 146, 149, 170]. Furthermore, this was also proposed in the German smart grids standardization roadmap [159] as well as in the "IEC Smart Grid Standardization Roadmap" [135]. However, not all aspects are covered by IEC 61499. First of all, its main intention was always to be used directly for the implementation of automation functions. Nonetheless, it is still equipped with a number of tools that can be used also for other modeling purposes (e.g., communication architecture). However, for use case or detailed architecture descriptions, IEC 61499 is not very well suited. Secondly, since IEC 61499 was not mainly intended for smart grids the gap between it and methods like SGAM or IEC 62559 is currently too big to allow for any affective automated approaches. In summary, IEC 61499 provides many modeling aspects that can help improving the smart grid engineering process. But in order to effectively use these, it needs to be better integrated with other smart grid approaches (e.g., SGAM, IEC 62559, CIM, IEC 61850). The next two sections provide an overview of the main aspects of IEC 61499, which are used in this thesis.

Overview of Main Concepts

The standard defines concepts and models that allow modular control software. To do this, the main modeling element of this automation approach is the so called Function Block (FB). Multiple FBs connected together into a FB network make up an IEC 61499 Application. The standard follows an application-centered modeling approach. This means that only after the complete Application has been defined it is deployed to field devices (called Devices in the standard) [64]. IEC 61499 specifies an architectural model in a generic way and extends the FB model of its predecessor IEC 61131-3 with an additional event handling mechanism. Due to this, FBs are an already established concept to define robust and reusable software components. They have a defined set of input and output parameters, which can be connected to other outputs and inputs, thus forming a complete automation application. In Fig. 2.4, an overview of the main IEC 61499 modeling elements is provided.

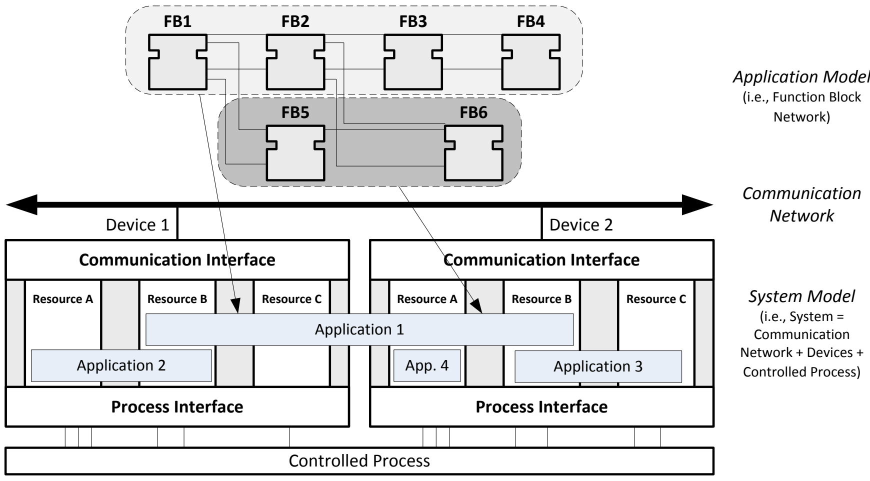


Figure 2.4: IEC 61499 reference models for distributed automation [64, 86].

One decisive difference between IEC 61499 and IEC 61131-3 is the execution model. As already explained IEC 61131-3 has a cyclic execution model. IEC 61499 has instead an event-based execution model. A consequence of this is that it also supports asynchronous execution. Thus, distributed IEC 61499 control applications can not only be executed in a synchronous way, through time triggered events, but also in an asynchronous way [86, 151].

Adapter Interfaces

IEC 61499 is mainly intended to be used as a graphical programming language. In large applications with many FBs and many connections readability and clarity may become a problem. The multiple connections often clutter the design space, which makes it difficult for the engineer to understand the interaction between the FBs. One possibility to overcome this problem is to use the Adapter concept provided in the IEC 61499 standard. With this concept data and events can be grouped together forming an interface. Adapters can be used in two different ways: either as an accepting interface, called Plug, or as a providing interface, called Socket [177]. Figure 2.5 shows how the Adapter concept is used in two composite FBs.

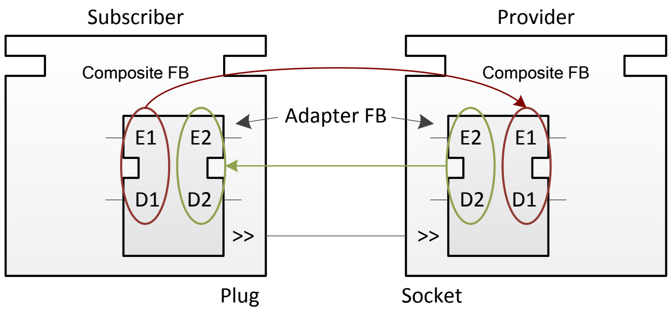


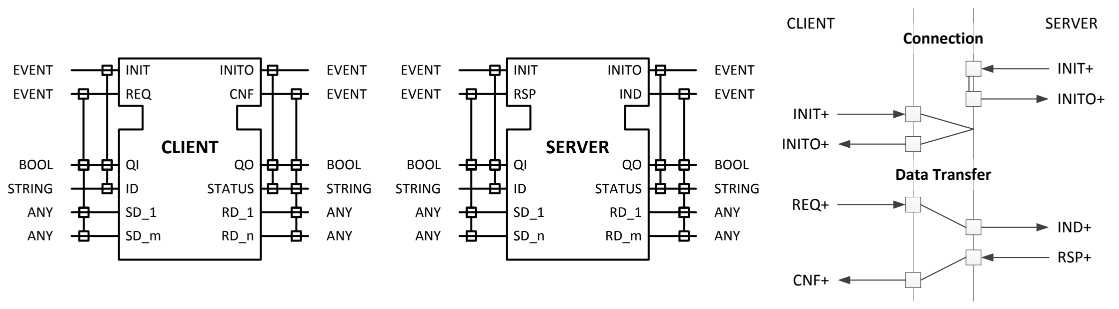
Figure 2.5: Adapter concept used in composite FBs according to [178].

The *subscriber* composite FB on the left uses the Adapter as a Plug. Inside the *subscriber* the Plug is represented as a FB. The Plug is connected to a Socket with the same Adapter type implemented by a *provider* FB. The Socket always has a mirrored interface of the Plug. In other words, events and data inputs of the Plug are outputs of the Socket, and vice versa for the Plug's outputs. Apart from reducing the number of connections, the Adapter concept also provides better decoupling of application parts [177].

Communication Patterns

Since the IEC 61499 standard has been developed for distributed control systems, communication is an important topic. This can also be seen in the system model, with interacting devices connected through a communication network (see Figure 2.4). However, as IEC 61499 is a generic standard it cannot provide specific communication means for this interaction. Nonetheless, it defines two generic communication models and also a suggestion for encoding and decoding IEC 61499 events and data according to the Abstract Syntax Notation One (ASN.1) specification. On one hand, bidirectional transactions are supported with a client/server model. On the other hand, unidirectional transactions are supported with a publish/subscribe model. These two models fulfill most requirements of distributed IEC 61499 applications. Since the two communication models are generic it is also possible to map other communication protocols that support either the client/server or the publish/subscribe approach [7].

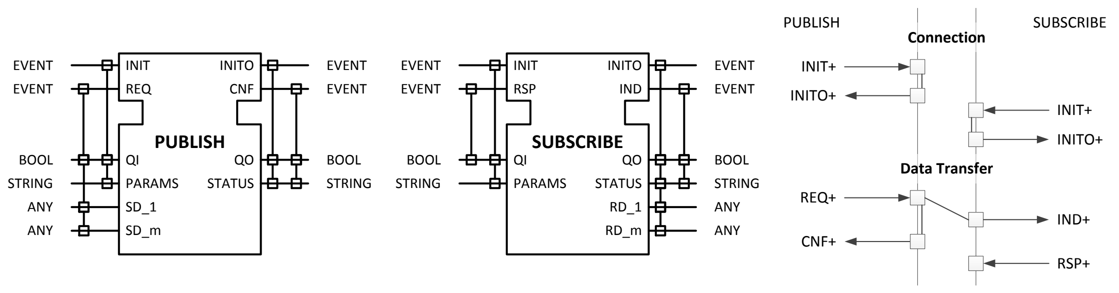
Generally, IEC 61499 uses so called Service Interface Function Blocks (SIFBs) to encapsulate communication services. For bidirectional transactions, a pair of SIFBs named *CLIENT* and *SERVER* is defined, as seen in Figure 2.6. Data can be exchanged from the client to the server and back again. Before any data can be exchanged both SIFBs need to be initialized. According to IEC 61499 the server needs to be initialized before the client's initialization, during which it establishes a connection to the server [64]. The typical usage scenarios for this model are master/slave interactions or remote service invocations (the client triggers a service provided by the server, which returns the result of the invoked service to the client) [7].



(a) (b)

Figure 2.6: Generic *CLIENT/SERVER* SIFBs for bidirectional transactions according to IEC 61499 [64]: (a) FB representation; (b) Sequence diagram.

The unidirectional method is described in IEC 61499 with a *PUBLISH/SUBSCRIBE* pair. Figure 2.7 shows interface definition of these SIFBs. Communication is directed from the publisher to the subscriber [64]. In contrast to the client/server model, the unidirectional model requires no specific order during the initialization. Furthermore, several subscribers may listen to the same publisher. Since the transaction is unidirectional, the publisher receives no feedback if data has been correctly received by the subscriber [7].



(a) (b)

Figure 2.7: Generic *PUBLISH/SUBSCRIBE* SIFBs for unidirectional transactions according to IEC 61499 [64]: (a) FB representation; (b) Sequence diagram.

2.3.3 General-Purpose Programming Languages

General-purpose programming languages are commonly defined as programming languages that are designed to be used in a wide variety of application domains. This usually means that the language does not have any constructs that are applicable for a certain application domain. Well known examples of general-purpose programming languages are Assembler, C, C++, Java, C#, JavaScript, and Python. Although they are all mostly used for different domains they are still general-purpose. Two main reasons why, for example, JavaScript is mainly used for programming webpages are, first of all, that it is supported by web browsers and, secondly, the many web-domain-specific libraries and Application Programming Interfaces (APIs) available for JavaScript [47].

Many automation solutions for smart grids are also programmed using general-purpose languages. Especially, for higher-level optimization and control functions. One of the main advantages with general-purpose programming languages is that they are very powerful. Any software problem in any application domain can be solved using them. Furthermore, this is also possible by learning only one or a few programming languages.

Despite these advantages, several issues can be observed for today's growing system complexity. These systems have advanced in complexity faster than the ability of general-purpose programming languages to mask it. Schmidt considers this one of the main problems with using general-purpose languages for cases where system-wide correctness is of high importance. He stated that developers "pay such close attention to numerous tactical imperative programming details that they often can't focus on strategic architectural issues such as system-wide correctness and performance" [128].

Another issue is timing. It is one of the things that is masked away by the many abstraction layers used in embedded systems design. However, when large complex, distributed, and connected systems are designed timing is a critical issue. When physical components are involved, where predictability is not 100 %, timing is always an issue. But, considering the fact that timing is not in the semantics of general-purpose programming languages, a program may execute correctly and still miss its deadline [84].

Smart grids is one domain where these mentioned issues can be observed. The smart grid is highly connected, complex, and the system-wide correctness is of the utmost importance. Accordingly, using only general-purpose programming languages for smart grid engineering may not be the best choice. Furthermore, in view of the discussion above general-purpose languages are mainly suitable for cases or system parts, where complexity and timing issues can be limited.

2.3.4 Information and Communication Technologies for Smart Grids

As mentioned above, the smart grid and the future power system is a highly interconnected and complex system. One of the key elements to handle such systems are appropriate ICT methods. The availability of ICT together with advanced automation and control concepts provide various opportunities to operate highly interconnected power grids with corresponding components in a much more effective way as in the past [145, 22].

In the dynamically developing field of ICT solutions for Smart Grids, the development of standards is of crucial importance mainly due to interoperability requirements. Currently, standards are mainly focused on non-distributed devices and systems since they originated prior to the increasing deployment of distributed generators. In order to ensure interoperability, international standards have to be taken into account. Several standardization organizations, various international projects, and roadmaps have analyzed this fact so far [138, 54]. In the following paragraphs, an overview of the most important ICT standards for smart grid systems is provided.

On international level, IEC plays an important role by providing common rules for the planning and operation of active power distribution grids. Especially, the IEC Technical Committees (TCs) TC 8 ("Systems aspects for electrical energy supply"), TC 13 ("Electrical energy measurement, tariff- and load control"), TC 57 ("Power systems management and associated information exchange"), and TC 65 ("Industrial-process measurement, control and automation") are responsible for smart grid related standards as mentioned in the "IEC Smart Grid Standardization Roadmap" [135]. This report suggests the following core standards to be used for the realization of Smart Grid related projects [135]:

- *IEC TR 62357:* Proposes a service-oriented reference architecture for EMS in the power transmission domain and for DMS in the power distribution domain.
- *IEC 61970/61968:* Introduces the domain ontology CIM for modeling the electrical grid and its components (see Section 2.2.3 for more information).
- *IEC 61850:* Covers the automation of substations and power utility equipment (see Section 2.3.5 for more information).
- *IEC 62351:* Describes security issues.
- *IEC 62056:* Provides data exchange rules and specifications for meter reading, tariff and load control in power systems.
- *IEC 61508:* Addresses functional safety rules for electrical, electronic and programmable electronic safety-related systems and devices.

In addition, the following standards also receive much attention, not only by the IEC roadmap, but also by other important smart grid related roadmaps [104, 159, 72, 123]:

- *IEC 60870-5:* Covers tele- and remote control protocols.
- *IEC 60870-6:* Responsible for inter-control center communication.

- *IEC TR 61334:* Addresses Distribution Line Message Service (DLMS).
- *IEC 61400-25:* Defines wind power communication rules.
- *IEC 61850-7-410:* Covers hydro energy communication issues.
- *IEC 61850-7-420:* Covers distribution energy communication issues.
- *IEC 61851:* Communication for EV, smart home, and E-mobility.
- *IEC 62051-54/58-59:* Defines metering-related standards.
- *IEC 62541:* Also known as OPC Unified Architecture (OPC UA), it provides a platform-independent information and communication model allowing to set up a Service-Oriented Architecture (SOA).

From these standards, it is clear that ICT is an important topic for smart grids. Also, these are only a selected extract of the available and used communication standards in the power system domain. As can be seen from this summary, it is important that a rapid engineering method can handle multiple communication and ICT standards. Furthermore, since these are always changing it must also be possible to incorporate new standards into the engineering method. From the mentioned standards above, IEC 61850 has received a lot of attention. This standard is described in more detail below.

2.3.5 Interoperability in Power Systems—IEC 61850

The first edition of the IEC 61850 standard was published between 2003 and 2005 under the title “Communication networks and systems in substations” [66]. Since then, parts of the standards have been revised and new editions have been released. The standard has also been extended to include automation outside substations. Accordingly, the standard has also been renamed to “Communication networks and systems for power utility automation”. The main goal is to increase interoperability between so called Intelligent Electronic Devices (IEDs) [66]. By standardizing the information used in power utility automation and providing it in an object oriented matter, interoperability between components from different vendors can be simplified. At the moment, the standard consists of 31 different parts and technical reports, but thematically the contents of IEC 61850 can be divided into three main topics: *modeling*, *communication*, and *configuration* [149]. These are illustrated in Figure 2.8.

The modeling topic of IEC 61850 covers a virtualization of physical components and functions into a formal data model. Logical Nodes (LNs) are the main modeling parts of IEC 61850. They are used to model power system components (e.g., switches, transformers, inverters) as well as power system functions (e.g., measurement, protection, voltage control). The LN-model defines semantics of real-time data that is exchanged between IEDs and is represented in text tables. The different elements of the model are hierarchically organized (top-down): IEDs, LDs, LNs, Data Objects (DOs), Common

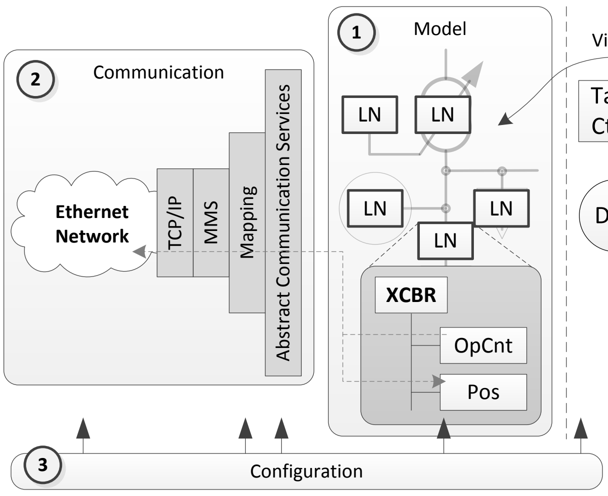


Figure 2.8: Overview of the main topics of IEC 61850: *(1)* modeling, *(2)* communication, and *(3)* configuration.

Data Classes (CDCs), and Data Attributes (DAs) [66, 9]. In other words, LNs are grouped together in LDs, which in turn are contained in IEDs.

The second topic of IEC 61850 describes how the information in the LN-model is communicated. First of all, it defines different communication services that are available to transfer this information between IEDs [66, 9]. This includes client/server-based but also publish/subscriber-based as well as real-time communication (i.e., Sampled Value, Generic Substation Event (GSE) and Generic Object Oriented Substation Event (GOOSE)). The use of real-time events enables a fast communication between control and protection devices, which can be used to improve system stability and performance [56]. Secondly, the standard also defines a mappings between the communication services and different protocols. For example, a mapping to Manufacturing Message Specification (MMS) is provided for client/server communication [66].

The third main topic of IEC 61850 is dedicated to the configuration of the first two topics. The main tool provided for this is the Extensible Markup Language (XML)-based System Configuration Language (SCL). As indicated in Figure 2.8, it can be used to configure different aspects. First of all, SCL includes artifacts to describe which components and functions are modeled, from a whole substation to a single IED. Secondly, it is also possible to configure how the information can be exchanged (i.e., which service and what protocol). Thirdly, SCL also provides tools to model and thus configure parts of the communication network [66]. The results are configuration files that can be used to configure the different devices in the system [18]. Figure 2.9 shows the main modeling objects provided by SCL.

Intelligent devices are one of the major preconditions for the realization of smart grids. In this context, IEC 61850 plays a major role for the standardized information and

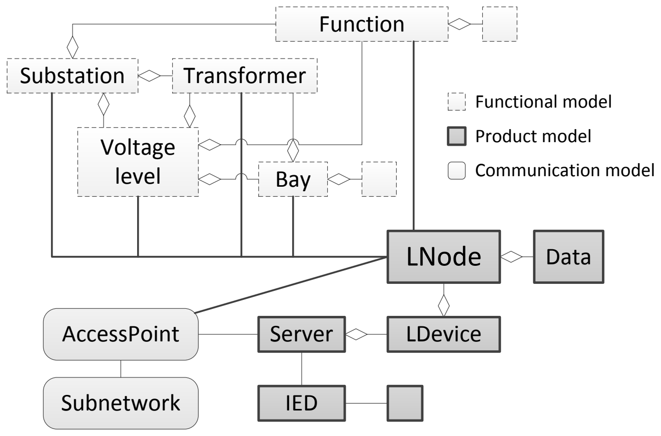


Figure 2.9: Main model objects as defined in IEC 61850-6. Adapted from [66].

data exchange. But, the main problem is that it only defines interfaces to the functions described by the LNs. The implementation of these functions and services is not covered by IEC 61850 but must be implemented using other solutions. Other standards or methods must be applied for defining this functionality (e.g. IEC 61131-3 or IEC 61499). Especially IEC 61499 has been proven to successfully integrate with IEC 61850 for smart grid applications [60, 170, 146]. Nonetheless, this topic still needs research.

Another issue with the IEC 61850 approach is its user-unfriendliness. First of all, the LN data model is only provided in text tables. Secondly, to find the complete definition of a LN, the user is required to switch between different parts of the standard in order to find all defined sub-types (i.e., CDCs and DA types). This is a very time consuming and erroneous process. Tool support is available for this process, but the tool suites are mostly commercial and/or proprietary products that are difficult to integrate into a rapid engineering method [59, 27]. Within the IEC TC 57, a UML representation of the LNs is also in development, but until now this has not been opened for the public [9].

2.4 Validation of Smart Grid Applications

Validation of smart grid systems can mainly be divided into three categories: simulative tests, laboratory tests, and pilot projects in the field. Whereas there exist a lot of literature about different general simulation methods, less can be found about validation standards for laboratory tests, and even less about methods for pilot projects. One reason for this may be that pilots are always designed for one specific use case, whereas simulation methods can be designed in a more general manner. Consequently, for laboratory tests the case is somewhere in between. The following sections provide an overview of general simulation methods for power systems and smart grids. Furthermore, an overview is also given about hardware and simulation methods for laboratory validation.

In summary, with the presented approaches below the validation of smart grid systems can be covered. However, these methods and tools usually focus only on the validation itself. Integration with other tools used for design and implementation is usually a challenge. Analyzing the below described approaches and concepts, it turns out that a major shortcoming is a missing holistic and standard-based environment for the design, implementation, testing and validating of smart grid applications. As it is not a goal of this thesis to develop any new validation methods one of the main focuses for the rapid engineering methodology will be to provide appropriate interfaces to current smart grid validation methods.

2.4.1 Traditional Power System Simulation

Power system simulations are traditionally used for a number of different tasks, including network planning, forecasting, transient stability, short-circuit analysis, power flow analysis, optimal load flow analysis, etc. Over the years a wide range of power system analysis tools have emerged. In general, the studies can be divided into two groups: steady state simulations and transient dynamic simulations [41].

Steady state simulations are often used to study how different modifications or changes impact the performance of the power system. To do this, the system is analyzed in a steady state. The focus is foremost to analyze if the power system variables (e.g., voltage or frequency) are within proper limits. In comparison, transient dynamic simulations are used to study how the power system behaves between stable states. Often the goal is to find out how the system behaves due to major changes, like switching activities or disturbances. In order to perform these studies, simulations need to include electromagnetic transients with a fine time granularity (e.g., micro- to milliseconds) [98]. Several commercial and open-source simulators support both types of problem analysis. Table 2.1 shows a list of common commercial and open source simulation tools [98, 41].

Table 2.1: List of traditional power system simulation tools [98, 41].

Simulator	Simulation type	Power system domain	License
Cymdist	steady state	generation, distribution	commercial
PowerFactory	transient, steady state	generation, transmission, distribution	commercial
EMTP-RV	transient	transmission, distribution	commercial
ETAP PSMS	transient, steady state	generation, transmission, distribution	commercial
EuroStag	steady state	generation, transmission, distribution	commercial
OpenDSS	transient, steady state	generation, distribution	open
PyPower	steady state	generation, distribution	open
GridLAB-D	steady state	generation, distribution	open
PSAT	steady state	transmission, distribution	open
PSCAD/EMTDC	transient	transmission, distribution	commercial
PSS®E	transient, steady state	generation, transmission	commercial
PSS®SINCAL	transient, steady state	generation, distribution	commercial

Most of the simulation tools in Table 2.1 have been used since many years and have proven their usefulness. Besides power system analysis capabilities, most of these tools also offer different interfaces and APIs. Using them, it is possible to control the simulation and also to exchange data (e.g., import/export of CIM models or database access) [98].

2.4.2 Simulation Methods for Smart Grids

The usage of simulations for validation and evaluation of complex systems has increased alongside the increased complexity of these systems (of systems). In general, different reasons may motivate the use of simulation tools and environments instead of studying the real system [124]:

- Analyzing a real system is too sophisticated/costly
- Tests within a real system are associated with a risk, economical or to infrastructure/personnel
- The real system is not existing (yet)
- Reproduction of experiments is not possible in the real system
- The real system is only poorly understood or very complex, respectively.

These reasons are not specific for energy systems but many of them also apply to the smart grid. As described above, traditional power systems simulation is mainly focusing on power system analysis and load flow calculation. However, classical power system analysis tools do not deal with the challenge of integrating existing models from other domains into an overall simulation, yet. For this purpose new solutions are needed.

The smart grid systems are characterized by a wide usage of ICT concepts. The combination of classic energy system together with ICT system calls for new smart grid specific simulation concepts. This is also driven by the need to determine the behavior and impact of new smart grid technologies such as DERs, energy storage solutions and different control mechanisms. Furthermore, the diversity of simulation approaches also leads to different categories that can be used to classify these works: "Single-Simulation Approaches" (i.e., multi-domain simulators) and "Co-Simulation Approaches" (i.e., coupling of several simulation environments) [124].

Approaches of the first category are characterized by the use of a single simulation tool to analyze multiple smart grid aspects. Complementary, other tools may be used for pre- or post-processing of data (e.g., analysis of the results), but the approach itself relies on a single simulation tool. Especially challenging with this approach is the combination of the two domains of telecommunication and power systems. However, for this case neither existing stand-alone power system analysis tools nor simulations tools for communication network simulation are sufficient to precisely model a fully interconnected power system with an ICT network [88]. Nonetheless, as mature tools exist for both of these domains, it is a natural approach to couple these tools into so-called co-simulations. It is an approach where models developed in different tools are jointly simulated through tool coupling. Each tool analyses one part of a modular problem. Intermediate results are exchanged between the tools during the simulation in discreet time intervals. Between these intervals the subsystems are solved independently [13]. From the functional point of view, a co-simulation is a simulation that is comprised of different models, each simulated by a different simulation tool.

In Table 2.2, a brief overview of simulation in traditional power systems and smart grids is provided. Basically, a broader scope and a much higher level of complexity must be handled by simulations for smart grids. In this case, it is important to not reinvent the wheel and use existing, established simulation tools and models [124].

Table 2.2: Comparison of simulation in traditional power systems and smart grids [124].

Traditional Power System	Smart Grid
monolithic	diversified
single system focus	multiple system focus
few control systems	comprehensive control strategies/mechanisms
simple scenarios	complex, large-scale scenarios

2.4.3 Hardware-In-the-Loop Validation

As discussed above real-world tests of new architectures and concepts for smart grid systems are not always possible or too costly. Pure simulation is one possibility to study such systems. One sub-domain of simulative validations is to use real-time simulations [39]. Especially, the real-time simulation approach in combination with real components, also known as Hardware-In-the-Loop (HIL), is a powerful evaluation method during the design and development phase. Such a validation setup usually provides a more realistic test scenario compared to pure software simulation [142]. Furthermore, the advantage of using a HIL approach has already been proven in various domains from the automotive, aerospace, and manufacturing domain testing automation and control functions [120, 75].

When the HIL approach is used in the power and energy domain, it is usually divided into the following two methods: Controller Hardware-In-the-Loop (CHIL) and Power Hardware-In-the-Loop (PHIL) [142]. CHIL normally refers to a HIL setup whereas the signals exchanged between the Hardware-under-Test (HuT) and the real-time simulation systems have a low power rating (i.e., low voltage and current) or through a digital communication connection. Such a configuration can be used to test and validate architectures, concepts and algorithms implemented in real controller devices together with a real-time simulation of a power system [172]. In comparison, the relatively new PHIL concept, which receives much interest from the power and energy community today, uses a power amplification device between the real-time simulator and the HuT. As a result of this power amplification the signals exchanged between the real-time system and the HuT have a higher power rating than in conventional HIL. This amplification is necessary to test power-electronic devices in a virtual environment [167, 4].

Both approaches are of great interest for smart grid application validation. They both achieve accurate results under realistic conditions with a relatively low financial effort. For both PHIL and CHIL, there exist complete validation environments provided by various manufacturers (e.g. National Instruments, Opal-RT, RTDS). They provide interfaces and computing power to perform HIL tests, often optimized for a certain domain [4].

2.5 Software Engineering Methods

With increasing complexity and an increased use of ICT technologies for smart grid applications, software engineering methods are gaining more and more interest also for this domain [41]. This section summarizes technologies from software engineering that are used or of interest for the work in this thesis. First of all MDE, is described in more detail. Secondly, a summary of component-based software engineering and how this is used is presented.

The presented methods below are also topics of research. However, it is not a goal of this thesis to participate in this research. Instead, these software methods are used as tools in order to analyze the main research question of this work. In addition, as the presented methods below are mainly intended to be used for software engineering, they need to be adapted in order to be used for smart grid application development. For this work, they are used as concepts and are refined for the current problem/domain (i.e., smart grids).

2.5.1 Model-Driven Engineering

As already mentioned and discussed in Section 2.3.3, increasing system complexity renders standard general-purpose programming techniques partly inadequate to ensure systems-wide correctness. One approach to handle this increasing complexity is to use MDE technologies [128]. In Section 1.2, MDE was already mentioned as a method that focuses on the development of models and the automatic transformation between these models. The main idea is to express designs using domain specific models. These provide abstractions in problem space, rather than abstractions of the solution space, and thus allows the engineer to solve a problem using concepts from a certain application domain (e.g., aerospace, biology, energy) [128].

In order to understand the idea MDE, its main concepts need to be explained. Summarizing, these can be divided into two groups: domain-specific modeling and model transformations. The following two sections describe these two concepts in more detail. Finally, a section devoted to Model-Driven Architecture (MDA) also summarizes how this approach builds upon the more general MDE approach.

Domain-Specific Models and Languages

The first and most central artifact of MDE is the use of models. In this case, models are representations of a system under study. The system may by virtual or a real-life system, either existing or intended to exist in the future [121]. They are said to be domain-specific since they are only used to represent a system from a certain application domain [128].

For such models to be effective, a method is needed for how to define them. For this purpose, MDE uses so called metamodels. Simply described, a metamodel is itself a model that can be used to describe a family of models [45]. Or, a metamodel defines a language for expressing a model [121]. A model, which is expressed using a language defined by a metamodel, is said to conform to its metamodel. Indeed, if the model is considered to be the system under study, it must follow that the metamodel must have its own metamodel. This metamodel is called a meta-metamodel. Of course this kind of abstraction can be repeated indefinitely. However, in practice four (meta-)modeling layers are usually used, as seen in Figure 2.10a.

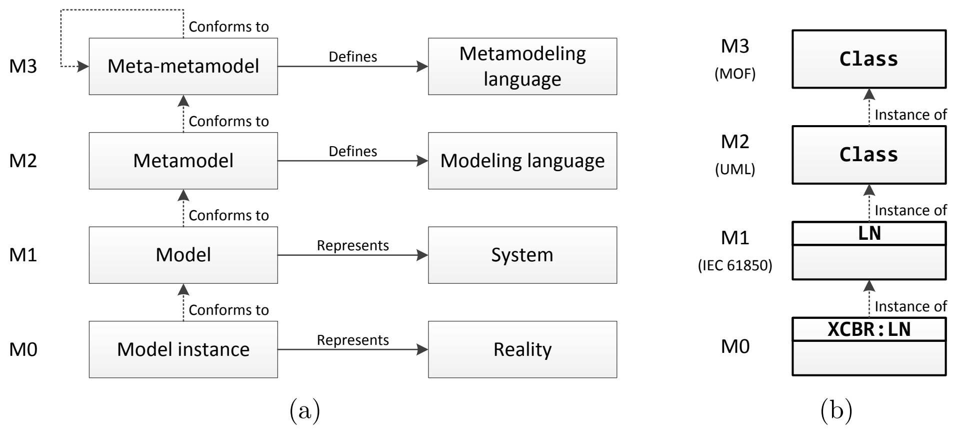


Figure 2.10: Metamodeling with four layers: (a) Metamodeling stack, based on [44]; (b) Example with a user model of IEC 61850.

The fourth layer meta-metamodel in Figure 2.10a conforms to itself. All other conform to their (meta-)model in the layer above. Complementary, Figure 2.10b also shows an example where a domain model of IEC 61850 has been created. With this, an instance of a LN, called *XCBR* is created. The LN is an instance of a *Class* defined by UML [164], which is one of the most prominent examples of metamodels [45]. In turn, UML conforms to Meta Object Facility (MOF) [97], which is one of the most well known examples of meta-metamodels [45].

To help the engineer in creating model instances, the concept of DSLs is often used [47, 108]. A DSL is a language which is especially defined to describe or solve problems of a certain domain. Basically, the idea is to allow domain experts to develop applications using a language with domain-specific notation instead of using general-purpose definitions.

One advantage of a domain-specific notation is that it increases the understanding of what a code implies. Consequently, errors are easier to detect, which also helps to improve the software quality [47]. In principle, either a graphical or a textual notation can be used. In both cases, the use of a secondary notation, the placement of graphical blocks or indention of text, is important to support the understanding. Whereas a graphical notation usually offers a lot of design freedom, text is more constrained and linear [110].

Model Transformations

The second main concept of MDE is the model transformation. Once a model instance is defined, the main idea is to transform it into other models or source code. A model transformation can be defined as an automatic generation of one or more target models from one or more source models. The heart of the model transformation is the transformation rule, which describes how one or more constructs of the source metamodel can be transformed into one or more constructs of the target metamodel. A set of transformation rules can be combined into a transformation specification describing how the source models are mapped to target models [30]. The basic concept of a model transformation is seen in Figure 2.11.

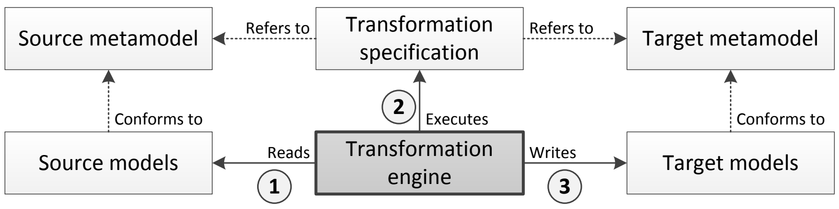


Figure 2.11: Basic concept of a model transformation [30].

To execute a transformation, the transformation engine reads the source models. In the next step it executes the transformation specification. Based on the rules in this specification, the last step is to create and write artifacts to the target model. Since the rules of the transformation specification refer to the metamodels the transformation only works if the source model also conforms to the source metamodel and the target model conforms to the target metamodel [30].

In general, the transformations can be grouped into model-to-model (M2M) transformations, model-to-text (M2T) transformations, and text-to-model (T2M) transformations. For M2M transformations, the result is also a model, whereas for M2T transformations—also called code generation—the result is plain text or source code. The last category has text as a source and the target is a model. Thus, all parsing methods are examples of M2T transformations. There exist numerous frameworks and languages to define such transformations. Some examples are Query View Transformation (QVT) [82], ATL Transformation Language (ATL) [78], and Henshin [12] for M2M transformations, Acceleo [1] and MOF Model to Text (MOFM2T) [101] for M2T transformations, and xText [36] for T2M transformations. Often multiple transformations are needed before an executable solution is achieved. In principle, the path to an executable system can follow one of two approaches. Either the final result is represented in code (i.e., created with a model-to-text transformation), or the other option is a model execution engine, which directly executes the model [100].

Perhaps needless to say, an MDE approach only reaches its full potential when the whole process is automated. In other words, after the engineer has defined the model instance the following transformations until the executable solution should be as automatic as possible. Automation also allows artifacts from models to be synthesized, thus helping to ensure consistency between implementations and requirements captured by models during the design phase. Schmidt states that this "automated transformation process is often referred to as *correct-by-construction*, as opposed to conventional handcrafted *construct-by-correction* software development processes that are tedious and error prone." [128].

Model-Driven Architecture

Although the MDE method describes in a general way how applications are created using a model-driven approach, there is still room for interpretation. Over the years different MDE initiatives have risen, where one of the most prominent is the MDA approach [96], developed by the Object Management Group (OMG). It has gained interest in computer science as one possibility to improve the software development process [132].

In summary, the MDA approach follows the general MDE methodology, adding additional architectural aspect on top. Its approach can be described by three main parts. The first step is to develop platform-independent application models using a Platform-Independent Model (PIM). In order to execute this model, it must first be transformed into a Platform-Specific Model (PSM). The PSM is the second main part of MDA and describes the application model in terms of a specific execution platform. The final main concept is the actual transformation of the PIM into the PSM [100]. One of the main differences compared to standard MDE is the platform-independence of MDA. This allows the developed application to be installed on different computing platforms using transformations from one PIM to multiple PSMs [121].

Together with the MDA initiative, OMG also suggest a number of other OMG standards to be used, namely: MOF for the meta-metamodeling, UML profiles to define DSLs—and thus also for the implementation of the PIM—and additionally QVT was specified as transformation language for transformations between PIM and PSM [121].

2.5.2 Component-Based Software Engineering

Component-Based Software Engineering (CBSE) is an established area of software engineering, which emphasis on decomposition of the system into functional or logical components. Each component should have well-defined interfaces used for communication between the components. Compared to object-oriented programming CBSE is considered to be an even higher level of abstraction. Different definitions of what a software component is can be found in literature. One of the early and most used definition was stated by Szyperski: “A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.” [156]. In Figure 2.12 a common representation of software components is illustrated.

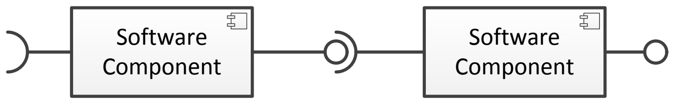


Figure 2.12: A common representation of software components, based on [29].

In order to define and construct software components, three main aspects are usually considered: functional specification of components, how to establish connections between components, and how information is exchanged. To describe the functionality of components, the interface specification is the most prominent part. An interface defines which actions/services are provided by a component and thus also accessible to requesters. A common characterization for an interface is a name and a number of parameters, which can either be inputs or outputs for the action [58]. One way to describe such interfaces is by means of an Interface Description Language (IDL). It is used to describe the interfaces in an implementation independent way (i.e., independent of a specific programming language). Through a mapping with an implementation language the IDL specification can be generated into code [29].

Accompanying the software component is often also an engineering process defining the component lifecycle. This process can be summarized by the following stages: modeling, implementation, packaging, and deployment [29]. The modeling stage contains the specification of the aspects described in the paragraph above. In the implementation stage this specification is produced into code. Sometimes only partial support for this stage is provided and it is intended that the component is implemented using standard programming languages. The packaging stage contains the preparation of the component before it is deployed. The final stage is the deployment of the components. It is the integration of the developed components into an executable system. In general, deployment is done at [29]:

- *Compilation time*: Components are integrated before system execution. Bindings are achieved through compilation and linking. Typically, the components together with the execution platform are combined into an executable image.
- *Runtime*: In this case, components are integrated into an already running system. This type of deployment is usually more complex since it requires additional information regarding the deployment of a component. Furthermore, this approach also requires management capabilities of the execution platform, which handles the integration of new components.

There exist multiple different models and frameworks that support CBSE. A number of common frameworks are among others: Common Object Request Broker Architecture (CORBA) [28], AUTOSAR [49], Open Services Gateway initiative (OSGi) [139], Java Beans [161], Microsoft Component Object Model (COM) [16], but also the previously introduced IEC 61131 (see Section 2.3.1) and IEC 61499 (see Section 2.3.2) count as component models [29].

2.6 Model-Driven Engineering for Smart Grid Application Development

There are advantages of using MDE technologies, especially for large and complex systems. This has already been proven in other work [128, 102, 137]. Furthermore, that the smart grid counts as such a complex system may also be seen as a fact [153, 145, 73]. As a result, it can be expected that MDE methods will also improve the engineering of smart grid applications. However, until now the use of MDE for this domain is limited. In general, the available work can be divided into three groups: architectural approaches, implementation approaches, and model mappings/transformations.

Previous work that concerns itself with architectural approaches tries to use MDE in order to improve the design, requirement and engineering process for smart grid applications. Consequently, these approaches have the most in common with this work. The usage of a model-based approach to understand, analyze and design smart grid systems was investigated by Lopes et al. [90]. Based on the NIST framework [53], they structure the design of smart grid system. Although they propose a model-driven approach, their solution does not directly focus on the actual implementation, but more on an understandable presentation of smart grid systems.

The previously mentioned SGAM Toolbox from Dänekas et al. also follows an MDA approach [31]. However the main attention of this work was more focused on the modeling process from business concepts to the PIM. How to transform the PIM into a PSM was not in focus. Nevertheless, thanks to the integrated code generation capabilities of Enterprise Architect, this option is also available for users of the SGAM Toolbox. However in this case, the generated code is not optimized for power utility applications and communication aspects are not covered. Knirsch et al. also based their work on the SGAM Toolbox [79]. They use the toolbox to implement a model-driven assessment of privacy issues based on data flows between actors. Secondly, the assessment is used during design time to study the impact it has on the modeled use case. As with the work from Dänekas et al., this work also does not present any solution for how to directly integrate the gained information into a possible implementation.

The second group of related work concerns how MDE can be used to improve the implementation of certain use cases. An MDE approach was presented by Paulo et al. already in 2005 [109, 43]. In this work, a meta-modeling architecture for design of substation automation systems was presented, as well as an object-oriented modeling language based on UML. This solution also supports control functions developed using IEC 61131-3 to be taken into account and allows to include them into the modeling. Compared to the architectural approaches described above, this work is more focused on implementation issues. Although code generation is provided to a certain extent this focuses on functional code. For example, the work does not consider a generic design of information exchange. Instead, this is assumed to be available, such as pre-configured IEC 61850 SCL files.

A similar case was presented by Lopez et al. [92], where they combine CIM and IEC 61850 using MDE methods for the implementation of smart substations. Another example is shown in the work by Yang et al. [174], where a method to automatically generate IEC 61499 applications based on IEC 61850 configurations is presented. In this case, it is expected that an IEC 61850 configuration is already available. Based on this, predefined FBs are generated to create an IEC 61499 application. Both cases are focusing mainly on the implementation issues. In other words, how to translate the PIM into a PSM. Compared to the architectural approach described above, these works do not directly cover the use case design and specification.

Work in the last category focuses on model mappings and transformation between models. For instance, work on the integration of CIM and IEC 61850 is a common topic. One example of a mapping between CIM and IEC 61850 is given by Pradeep et al. [113]. They propose a direct mapping between objects in CIM and IEC 61850. Another model-driven approach using CIM, IEC 61850 and Companion Specification for Energy Metering (COSEM) is proposed by Brédillet et al. [17]. Although this work does not describe any details on the implementation, it shows a concept of including different standards into a model-based approach. Santodomingo et al. present an ontology matching approach, which is used to find matches between CIM and IEC 61850 [126]. In this work, a translation method between CIM configuration files and configuration files for IEC 61850 is presented. Another approach by McMorran et al. presents a method for automatic transformation from/to CIM [94]. The idea is to provide a better support for import/export of CIM models to power system analysis tools from different manufacturers. With the smart grid engineering process (i.e., with design, implementation, validation, and deployment) in mind, the work in this last category focuses only on a small part of the process. Although it provides solutions for specific model transformations, it does not provide a general solution for smart grid application engineering.

Analyzing the methods and approaches presented above, and in the previous sections, a number of missing features can be identified. Consequently, these unsolved questions also serve as a basis for what must be achievable by the rapid engineering methodology developed in this work. The following list provide a summary of the most important missing features:

- *Holistic engineering:* The main drawback of existing approaches up to now is that non of them cover the whole development process in an integrated manner.

No approach is available that combines design, implementation, validation, and deployment in one methodology.

- *Model-based:* Model-based engineering concepts for smart grids are missing or only partly available. The existing approaches only cover one part of the development process are mostly focusing on single aspects.
- *Effortless transitions:* At the moment there is no method that completely removes the effort of moving from one engineering step to the next. Methods that provide automation mostly focuses their efforts on the work within one step.
- *System of systems:* Current engineering methods are primarily focusing on the development of single systems and not a system of systems (e.g., one focus on the development of substation automation). Consequently, there are no methods available for modeling and design of the overall application.
- *Multiple domains:* Current approaches are concentrating on handling one domain. No approach is currently available which integrates domain knowledge from different domains, such as control, communication, or power system modeling.
- *Integration with legacy systems:* Many of the current approaches do not offer any support for the integration of legacy systems, be it functionality or communication configurations.
- *Smart grid domain specific:* Approaches that are optimized for smart grid engineering are only partly available. Many methods can be found for other domains, but these are often not directly usable for smart grids.

Summarizing, up to now there is no integrated approach available that covers the whole development process—from use cases design to an executable implementation—using an MDE approach that can handle the multi-domain aspect of smart grids. Therefore, it is the purpose of this work to provide the concept for such a solution. Consequently, the goal of this work is to show that such an holistic approach is possible and thus to fill the gaps that are still existing for rapid engineering of smart grid applications.

CHAPTER 3

Requirements for Smart Grid Automation

Before a rapid engineering method can be conceptualized, requirements for the engineering process must be identified. But, since the engineering process can differ very much between companies, stakeholders, and even users, it is important to first identify for whom the rapid engineering method is intended. In this thesis, focus will mainly be on the needs of companies and stakeholders, generally described as business actors.

First of all, this chapter presents a number of business cases that have the purpose of highlighting different engineering problems—and possible solutions—for different business actors. For each business case, selected use cases will also be presented. Based on the business cases and the example use cases, the next step is to identify requirements for the engineering methodology.

3.1 Selected Business Cases

Three main business cases were chosen. They are based on experience from smart grid research projects as well as use cases collected from repositories [38]. The main goal of the business cases is to show the range of engineering problems that stakeholders are currently faced with for smart grid development. These business cases will be used as a basis to identify business actors and finally requirements for the engineering methodology.

3.1.1 BC1: Use Case Design

This business case is intended to show the usefulness of structured use case design and specification. Furthermore, if these specifications can be directly used for implementation, preferably in an automated way, a much more rapid and cost effective implementation can be achieved. This business case is mainly motivated by activities in a number of recent research projects, such as IntelliGrid [62], DISCERN [127], FP7 ELECTRA IRP [10], H2020 ERIGrid [153], and IEA ISGAN/SIRFN [134]. In these projects, the design and also the implementation as well as the validation of smart grid use cases are core topics.

Problem Definition

There is an increasing usage of structured use case description methods (e.g., SGAM [25], IEC 62559 [69]). These methods have in common that they provide a (textual) template for specifying use cases. An advantage of using a template is that it simplifies the comparison of different use cases. Another advantage is that it is easier to find specific information (e.g., actors, information exchanged) if a structured description is used. Template based use case description methods are often used to specify high-level use cases. High-level since they usually do not provide any detail about the implementation of the use case. Nevertheless these specification are an important step in the engineering process for smart grid applications.

Structured use case descriptions are an important step in smart grid application engineering. However, until now the information provided in the use case cannot be directly used as an input for the implementation. Normally the engineer has to read the use case description and create the implementation based on her/his interpretation of it. This process is critical for the outcome of the whole engineering process, and it is very prone to human errors. Furthermore, it is a time consuming process, especially if the use case description is unclear or vague.

Recommendation

The provision of a methodology for how the information from the use case description can be directly used in the implementation process would improve the quality of the final product. Furthermore, if the use case information can be understood by an engineering tool it can be used to automatically generate the final implementation, or at least parts of it. In order for this to be possible, the use case information must be available in a machine readable and formal way. With an automated approach from use case specification to implementation, the following aspects can be utilized:

- Functionality can be automatically generated based on the functionality descriptions in the use case specification
- Interfaces to other systems or actors can be automatically created based on the information flow descriptions of the use case
- Round trip engineering can be utilized, which will allow automatic changes to the use case descriptions in case of later changes in the implementation.

Justification and Anticipated Outcomes

With the proposed automatic generation of the implementation from the use case description a number of advantages is achieved. First of all, it is anticipated that the final implementation will be less prone to errors. Due to misunderstandings or ambiguous information, the resulting realization may have missing or erroneous features. By providing a formal method for use case specification, where the information is machine readable, the use case information will be more intelligible. Furthermore, with an automated generation of implementation parts, human errors can be decreased. Often information that is needed for a correct implementation may not be available in the use case specification. In other cases, the engineer may misinterpret the use case information and thus the resulting solution may not be that what was ordered.

It is also expected that the time to market can be significantly reduced. With the traditional approach, the same work is often done multiple times: first during the use case design and later again during the implementation phase. By providing an automated generation of much of the implementation, this time can be decreased. Furthermore, due to the anticipated lower amount of errors, less time will be spent with error correction.

In total, due to fewer errors and a shorter time to market, lower costs are expected.

Example Use Case

This use case is an example of how a template based use case description methodology can be used. It was collected from the EPRI Use Case Repository [38] and describes a central volt-VAr controller, using the IEC 62559 template [52]. The use case is typical for distribution network operation, where a central system manages and controls all *Volt-VAr Controller (VVC)* devices in a regional distribution network. The use case description is available in Appendix B.1.

The idea with this example is to show how use cases are described according to the IEC 62599 use case template. From this, it will become clear that only providing a structured approach is not enough in order to generate implementation code. Still, the benefit with a structured template that also allows the user a certain amount of freedom must not be lost with the automated solution proposed above.

Relevance for the Engineering Methodology

High-level use case descriptions are an important step in smart grid application engineering. This is often the first real attempt at specifying the use case functionality in a structured way. Especially the IEC 62559 use case template and the SGAM approach have proven their usefulness in this regard. The more complex the use case becomes the greater the benefit from a high-level description. Consequently, formal design and specification of use cases, compatible with IEC 62559 and SGAM, is one important requirement for the engineering method presented in this thesis.

Furthermore, as stated by this business case the possibility to automatically generate implementation artifacts directly from the specification is a big advantage. Therefore, the rapid engineering method should provide such possibilities. With a formal method for specification, combined with rules stating how to generate implementation artifacts, the basis for an automatic generation is provided.

This business case is relevant for many smart grid stakeholders. The more comprehensive the use case, especially in terms of information flows between different actors, the more useful is a detailed use case description. Thus, utility operators are typical owners of this business case. But, the automatic engineering process may indeed be interesting for others as well, including system integrators, device vendors, or plant operators.

3.1.2 BC2: Utility Operator Control Implementation

The goal of the project "DG DemoNet—Smart LV Grid" [19] was to find solutions for an active network operation for low voltage networks. Two of the main goals for the project were to find monitoring and control approaches in order to facilitate the system integration of DERs and electric mobility. The objective was to integrate these with acceptable costs regarding investment, maintenance and operation. Both solutions for central and distributed control concepts were tested in three pilot regions in Austria withing the project [81].

This project is a typical example of a control and grid support application where a utility operator has to integrate and combine different resources, both internal and external. The power system is a distributed system, both from a geographical point of view, but also from a stakeholder point of view. Thus, since the utility operator cannot control what products are used by the other stakeholders, the interoperability between components from different vendors becomes even more important.

Problem Definition

The changes towards a smart grid requires the DSO to constantly update its grid support functions. During the development of control and grid support applications for distribution networks there is often the need to include resources or ancillary services from grid components. This can be components owned by the DSO, especially intended for grid support, or by other stakeholders (e.g., a PV inverter). Even for the components owned by the DSO, the engineering and configuration can be a challenge, especially due to the geographical dispersion. But, even more challenging is the engineering and configuration of components owned by other stakeholders. For this business case, two possibilities are studied: functional and communication configurations.

Through the introduction of ancillary services provided by DERs, it has also become necessary to provide configuration possibilities for these services. The number of available ancillary services has increased in the last years and it is not always possible to use all at the same time. For example, setting the power factor in parallel with the active and reactive power of an inverter will very likely lead to conflicts. Therefore it must be possible for a DSO to configure what ancillary services should be active at a certain moment. However, this requires the DSO to define which services are available by the DER and how to configure them.

One aspect that is becoming more and more complex is the configuration of communication interfaces and information flows. With an increasing number of distributed components that need to be controlled and monitored there is also an increasing need for unified configuration possibilities. Software Defined Networking (SDN) is one approach to handle switch configurations in a centralized way. But, this only solves routing on a lower layer and is not data dependent. However, with the increasing introduction of comprehensive communication protocols and information models (e.g., IEC 61850, OPC UA) it is now also possible to configure what information is exchanged between components.

Due to the high number of different stakeholders, combined with an even higher number of different component vendors, the process for communication configuration is complex. Furthermore, the amount of different protocols that need to be supported makes the situation even more complex. At the moment, there is no unified solution to handle communication configuration. Neither is there a solution for configuration of different communication protocols, nor a solution that is completely vendor independent. As a result, a very time consuming and costly configuration process is currently needed.

Recommendation

It is the recommendation to provide a process and methodology for the DSO that unifies the functional and the communication configuration. This should also be indifferent of the communication protocol or the vendor. The process should allow representation of available services and communication capabilities provided by DERs. It must also support the DSO to change these in order to fit the current application. In order to effectively help the engineer, the process should be automated.

In IEC 61850, a process is defined for the configuration of IEDs. The process is shown in Figure 3.1 and is described by a number of steps: *(1)* a vendor provides a template with the data model and general capabilities of the IED; *(2)* the template is imported by the system configurator where instances are created, containing system and information flow specifications; *(3)* the IED specification extended with the system information is provided to the IED configurator; *(4)* a final version of the IED configuration is downloaded to the IED; *(5)* it is also possible for the system configurator to directly obtain the current IED instance from the IED configurator [66].

The process in Figure 3.1 was only intended for IEC 61850, but can also be extended for functionality and other communication protocols. In that case, templates from different functions and protocols, from different vendors, must be supported. The system configuration must also be general enough to handle different information models. The following points summarize the main solution:

- A general data model should be created that supports representation of services,

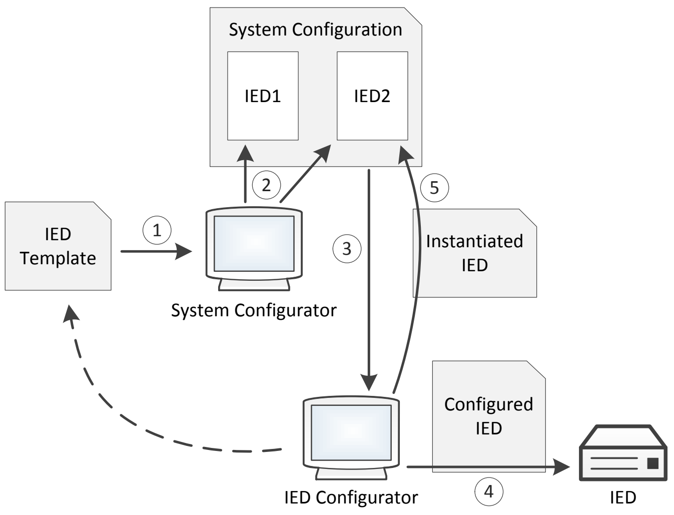


Figure 3.1: Configuration and modification process for IEDs according to IEC 61850 [66].

different protocols, and information models (covers step *(2)* and the system configuration in Figure 3.1).

- It should be possible to import templates of functionality or protocol descriptions from different vendors, and to integrate them with the system configuration (steps *(1)* and *(5)* in Figure 3.1).
- Based on a general system configuration, it should be possible to create configurations for specific IEDs or DERs, configuring their functionality and communication interfaces (step *(3)* in Figure 3.1).

Step *(4)* is considered as proprietary by the different vendors and is not covered by the recommendation.

Justification and Anticipated Outcomes

With the proposed solution, it is expected that the process for functional and communication configuration is simplified. This hopefully also means that less time is needed and that the cost for configuration can be reduced. In general, a number of advantages can be achieved. As with BC1, it can also be expected that errors are reduced. By automating much of the process in Figure 3.1, there is less risk that human errors are introduced.

Another advantage is that switching between protocols will be much simpler. With an independent data model used in the system configuration, the information description is also independent of the actual protocol that should be used. Consequently, the actual protocol that should be used is not decided until step *(3)* in Figure 3.1. This also means that if another protocol should be used only step *(3)* needs to be repeated.

Example Use Case

This section gives an example of an implementation use case that would benefit from the proposed solution. In Appendix B.2, the example is also provided as a use case description according to IEC 62559. The use case is taken from the project “DG DemoNet—Smart LV Grid” [19], where it was the goal to find solutions for an active network operation for low voltage networks. These solutions were mainly focusing on enhancing the voltage quality in the grid, with the help of DERs and electric vehicles. In the end several control approaches were suggested. Also within the project, the control approaches were tested in three pilot regions in Austria [81].

The developed control concepts follow a step-by-step approach. In total, four stages were designed for the control concept, and implemented as a low voltage grid controller. The controller is an industrial Personal Computer (PC) and is located in the secondary substation, where it can access voltage measurements from smart meters in the field. The four control stages can be summarized as follows [81]:

- *Stage 1 - Local Control*: In this mode, the local actuators only act on local measurements, and there is no communication between components. The main actuator is the MV/LV transformer equipped with an on-load-tap-changer. Additionally, PV inverters and EV charging stations are controlling the voltage locally using droop curves for reactive, Q(U), and active power, P(U).
- *Stage 2 - Distributed Control*: Here, a measurement and communication infrastructure is used. Voltage values from critical nodes in the grid are measured and transmitted to the central controller. The controller uses this information to find an optimal tap position of the transformer. Additionally, the PV inverters and charging stations from Stage 1 are still in droop control mode.
- *Stage 3 - Coordinated Control*: Additionally to Stage 1 and Stage 2 the coordinated control also updates the predefined Q(U) droop curves of the PV inverters and the active power droop curves charging stations. These updates are sent via broadcasts messages from the central controller.
- *Stage 4 - Selective Coordinated Control*: This stage is essentially the same as Stage 3 with one exception: updates of the droop curves are only sent to specific inverters and charging stations instead of being broadcasted to all remote units.

For the implementation of the control concept, a number of adaptations to the original system were needed. Functionality was added at substation level through the developed low voltage controller. Also, the involved PV inverters were extended with new functionality to implement the necessary droop curves. The involved components were provided with necessary communication interfaces and were connected with each other through a PLC network. Figure 3.2 shows the overall ICT structure of the low voltage grid control concept. It also shows the communication protocols that were used [19].

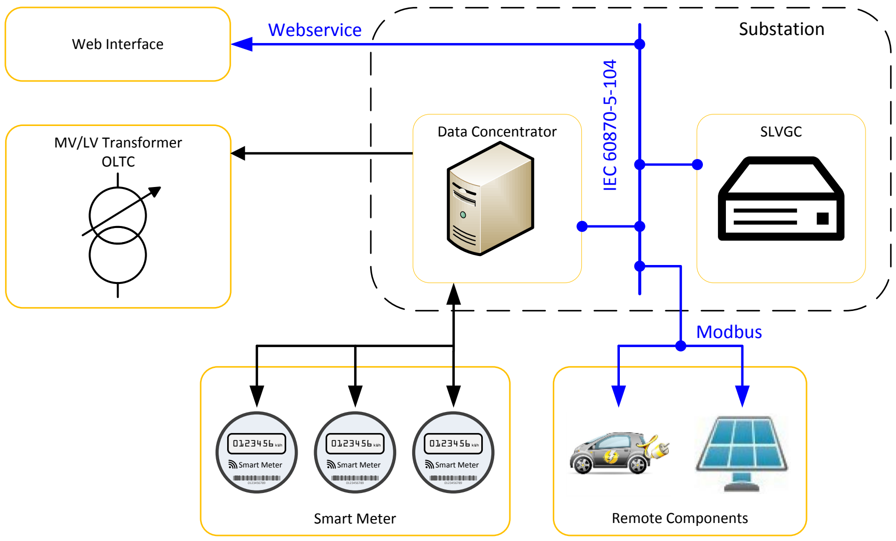


Figure 3.2: Hardware setup of the low voltage grid control concept. Adapted from [19].

At substation level two main components are seen: a data concentrator and the industrial PC running the Smart Low Voltage Grid Controller (SLVGC). The data concentrator is responsible for the communication to the outside world. This includes the MV/LV transformer, smart meters, and the remote inverters and charging stations. From the data concentrator the controller receives the state of the grid through a number of measurements. Part of the project was the implementation of the SLVGC and the extensions to the PV inverters [19], indicated by the PV array in Figure 3.2.

Relevance for the Engineering Methodology

The increasing number of DERs, and the possibility to use their ancillary services for grid support, are resulting in more and more distributed control applications. As shown by this business case the possibility to configure functionality and communication of DERs and IEDs can highly improve the implementation process for complex control and grid support applications. Consequently, this is also something that must be supported by the rapid engineering methodology developed in this thesis.

The use case is one example that would benefit from an automated configuration possibility. When multiple components are used together with an ICT network, there will always be a need for configuration. With the proposed solution, this can be done platform independently. This allows the engineers to put more focus on the control functionality.

This business case is mainly applicable for utility operators and system integrators since they are often commissioned by utility operators for this kind of projects. Furthermore, on a lower scale it may also be of interest for plant operators, especially for larger plants or virtual power plants. They are also dependent on ICT systems and ancillary services must be divided over multiple DERs.

3.1.3 BC3: Ancillary Services from Component Manufacturers

The increase of renewable energies and Distributed Generations (DGs) in recent years creates new problems and challenges for grid operators. As a consequence, ancillary services for both medium and low-voltage networks have received a higher focus in projects as well as national grid interconnection codes [111], [157]. Furthermore, as seen in a number of research projects and in national regulations, direct market participation by smaller DERs is increasing [77, 99, 119].

This business case highlights some of the challenges encountered by component manufacturers during the development of extended services of their components. The business case is motivated by the OpenNES project [118], where it was one of the goals to improve the development process for DER manufacturers.

Problem Definition

Since PV generation is almost exclusively connected at distribution level, and most of it at low-voltage level, local problems, and in particular voltage rise problems, are increasingly experienced in rural areas of several countries. Also in this case, ancillary services (local voltage control) can help maintain the quality of supply, and also increase the network hosting capacity [141]. Many existing grid codes already require DERs to be able to provide ancillary services (e.g., for voltage control purposes) [157, 23, 114, 160].

Even if the main types of ancillary services are generally specified, many grid codes still do not specify any particular requirement on the implementation of these services (e.g., the dynamic response of the controls [114, 160]). In other cases, the specifications are more detailed, but still written in a generic way. For example the German guideline [157] as well as the Italian standard [23] both have rather general requirements on the implementation of the local voltage control support. The Italian standard requires the maximum response time to be lower than 10 s and the German guideline states a variable time between 10 s and 1 min, specified by the grid operator. The varying specifications between different countries increase the complexity of developing ancillary services. As a consequence two different implementations may still be required although the same ancillary service is specified by two different countries. This problem is also increasing, since the required ancillary services may change rapidly, due to unforeseen problems.

One such unforeseen problem is the so-called “50.2 Hz problem” in Germany. It was caused by the automatic disconnection of PV systems in case of non-severe over-frequencies. But with a PV generation capacity exceeding 30 GW, the coincident disconnection of all PV systems only made the situation worse. Hence, the system safety was not guaranteed anymore and corrective actions had to be taken [50]. The solution was to change how inverter based generation systems react to over-frequencies in the applicable grid code. Instead of directly disconnecting, the active power is linearly curtailed when the frequency rises above 50.2 Hz [111]. The 50.2 Hz problem even required already installed inverters to be retrofitted. This can be a very costly operation if no remote access and deployment possibilities exist.

The energy market is also changing, as a result of the increasing amount of DERs [77, 99]. Since 2014, all DERs in Germany with a peak power production above 100 kW must participate directly in the energy market [119]. Although not mandatory, this possibility also exists for smaller power. This opens up new possibilities for virtual power plants and other services. However, it also introduces new technological challenges for manufacturers, since they need to provide the necessary ICT interfaces to enable the market participation. Furthermore, if the new services are not part of their core business model it increases the willingness of manufacturers to use third-party service providers. But, for this to be possible it must also be allowed by the used engineering process.

Recommendation

Based on the formal specification method that was recommended in BC1, a number of extensions can be made in order to improve the engineering method of ancillary services. The different country specifications often have the same ancillary services, but the implementation may vary. Therefore, it should be possible to create template functionality that can be adapted for a certain use case. Many ancillary services follow the same pattern. For example, a linear relationship between input and output (e.g., reactive power output based on the voltage input, or active power output based on the frequency input). Such functions can all use the same template.

It is also proposed that functionality is developed independently of the execution platform. Since various countries may require different hardware platforms the same ancillary service may need to run on different platforms. Platform independent development increases the usability since it is not necessary to re-implement a function only because the platform changes. The 50.2 Hz problem also shows that remote deployment of functionality is important. Platform independent function development is a prerequisite for this, but the engineering method must also allow that functions are installed to remote DERs.

It is also recommended that the engineering method allows simple integration of functionality developed by third-party service providers. This allows the component manufacturers to outsource the development of services where they do not have the time or enough in-house knowledge.

Justification and Anticipated Outcomes

As with BC1 and BC2, it is also for this business case anticipated that the proposed solution will reduce the engineering effort. The implementation will be simpler and more structured. This will reduce the risk for human errors and decrease the time needed for the implementation.

Often new grid code requirements only apply to new installations. Thus, currently installed systems and components are not affected. However, when critical changes are made (as with the 50.2 Hz problem in Germany [50]), even existing systems may need to be retrofitted. With the proposed solution, manufacturers will support the possibility to remotely change these installed systems, through updates or reconfiguration.

The direct market participation is one case where additional services are added to a DER component. In principle, the connection to a market interface can be implemented directly by the DER manufacturer. However as the number of services increases, so will also the complexity and needed development resources from the manufacturer. Instead of doing all development in-house, manufacturers can outsource the implementation of certain services (e.g., the direct market participation).

Generally, the engineering solutions proposed in these three business cases are all intended to reduce the human efforts and increase the quality of the implementation. By reducing the manual effort, time can be saved, which also means reduction of the engineering costs. The same applies to implementation quality. Artifacts that are created through an automatic generation process are often referred to as "correct-by-construction". This can be compared to the traditional development process where testing is often tedious and error prone [128].

Example Use Case

As an example to show the development of ancillary services, the German MV guidelines [157] specify how power plants should be able to provide reactive power for grid support. The guideline states that with an active power output, either a fixed reactive power provision is used or the network operator can adjust the target value through remote control. The reactive power setting can be adjusted through one of the following:

- Fixed active power factor $\cos\phi$
- Active power factor $\cos\phi(P)$
- Fixed reactive power in MVAr
- Reactive power/voltage characteristic Q(U)

Figure 3.3 shows an example of a Q(U) characteristics. The guideline also states that for the Q(U) characteristics it should be possible for the network operator to adjust the reaction time between 10 s and 1 min. Furthermore, to avoid voltage jumps, it is advisable to choose a characteristic with continuous profile and limited gradient. In Appendix B.3, a use case description according to IEC 62559 is provided. It describes the functionality of a Q(U) control for a DER.

Relevance for the Engineering Methodology

The engineering required in order to comply with grid codes from different countries is for many manufacturers time consuming and challenging. Although many grid codes have roughly the same requirements there are still many exceptions and differences. First of all, not all grid codes require the same ancillary services to be implemented. Secondly, even if the same ancillary service is required it may still be defined differently. This

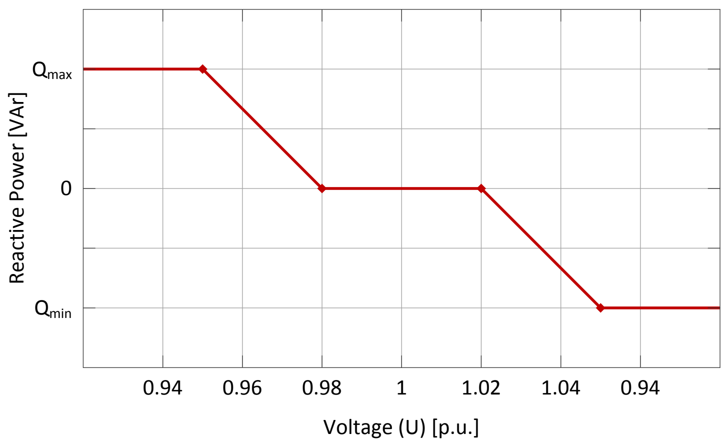


Figure 3.3: Example of a Q(U) characteristic.

means that the rapid engineering methodology must be flexible enough to handle different requirements depending on the country of sale.

Furthermore, the proposed solution of this business case leads to a number of requirements regarding the implementation of functionality. These must be supported by the rapid engineering method:

- Template functionality or configurable software components should be supported.
- Platform independent development of functions should be supported.
- Deployment of functionality to a remote component should be supported without the need for an engineer on-site.
- Integration of third-party services should be supported, to outsource the development of single functions to third-party service providers.

This business case was developed with a component manufacturer in mind, especially a DER or an inverter manufacturer. However, the proposed solution would certainly benefit other stakeholders as well (e.g., utilty operators, system integrators, third-party service providers).

3.2 Business Actors and Requirements

Based on the business cases from Section 3.1, actors and requirements can be identified. Together with the overall research question and the goals of this thesis from Section 1.2 they formulate the main foundations for this work.

3.2.1 Business Actors and Stakeholders

From the business cases a number of actors can be identified. These are stakeholders that have an economical or practical interest in the engineering of smart grid applications. Not all of them are actors in the sense that they interact directly with the rapid engineering methodology, but all of them have an interest in the outcome. Table 3.1 shows the identified actors without any specific order.

Table 3.1: Actors and stakeholders involved in the rapid engineering methodology.

#	Actor *name* and description
A1	*Utility Operator* The operator of the power grid. This can for example by either the DSO or the Transmission System Operator (TSO). A common activity for the utility operator is the specification of a certain use case or functionality before implementation, see BC1. But the utility operator may also be directly involved in implementation and validation of applications, as in BC2.
A2	*System Integrator* The system integrator delivers and integrates a whole or part of a system (e.g., a substation) to a client, which is often a utility operator. The system integrator may have their own components or they are purchased from another manufacturer/device vendor. A common setup is that the utility operator specifies a certain application (according to BC1) that is implemented by the system integrator (e.g., BC2).
A3	*Manufacturer/Device Vendor* The manufacturer of grid components. This actor must have the possibility to implement functionality on all security levels of a component (i.e., low-level as well as high-level functionality). This actor is active in both BC2 and BC3.
A4	*Third-Party Service Provider* The third-party service provider may be interested in implementing services for smart grid components. One example is the implementation of direct marketing services for smart DERs, see BC3.
A5	*Plant Operator* This actor is the operator of a power plant or a flexible load (e.g., a building or an energy storage unit). This could for example be a virtual power plant operator who needs to optimize the usage of the involved plants. It may also be an aggregator for ancillary services or flexibility. The plant operator may also be interested in implementing additional services for their plants, see BC3, or overall optimization algorithms, see BC2.

A6 *Design Engineer*
The purpose of the design engineer is to develop conceptual designs and specifications that ensure the correct functionality of a product. Furthermore, this is often done in order to meet customer and company needs. In some cases the design engineer also is responsible for prototyping. This means that a model or prototype of a product is created and tested.

A7 *Implementation Engineer*
Based on the specifications and the initial design made by the design engineer, the implementation engineer is responsible for creating a functional implementation of the product. For software products, this would imply creating a functional copy of the software for a certain platform. The implementation engineer is involved in the development of use case for BC2 and BC3.

A8 *Test Engineer*
Once an implementation is available the test engineer is responsible for evaluating if the product fulfills its requirements. This can partly be done through software tests, but in the case of smart grid applications with interacting actors also simulative and laboratory studies should be made. With these methods, it is also possible to study the behavior of the product interacting with other components. The test engineer is involved in the testing of developed use case for BC2 and BC3.

A9 *Deployment Engineer*
The deployment engineer takes over after the product has been successfully validated. It is her/his responsibility to coordinate all activities related to ensuring the correct deployment and proper installation of the developed product. This must be done in cooperation with the product owner. For example, in BC2 the product owner would be the DSO.

3.2.2 Requirements

It is the intention that the rapid engineering method should support the proposed solutions presented in the business cases in Section 3.1. Therefore, these are used as a basis for identifying requirements for the rapid engineering methodology. In Table 3.2, the result of this identification is presented.

The requirements are inspired by the business cases, but are also more general compared to the recommendations presented in the business cases. The rapid engineering method should provide solutions for the business cases, but it should also be more than that. Furthermore, a number of requirements are identified that are more related to the goals of this thesis. Thus, in total, the requirements in Table 3.2 provide a framework, with boundaries, which in the end will contain the rapid engineering methodology.

Each requirement in Table 3.2 is assigned a priority. This is not directly related to the priority of the requirement for the stakeholders. Even if a requirement has a low priority in Table 3.2 it does not mean that all of the stakeholders also do not find it important. Instead, the priority is intended to prioritize the work in this thesis. Thus, during the development of the rapid engineering methodology, more focus is put on requirements with a higher priority.

Table 3.2: Requirements for the rapid engineering process.

#	Requirement *name* and description
R1	*Design and Specification* Priority high The specification and use case design phase is the first attempt at defining the behavior of an application. During this phase, user specifications are collected and formalized. Compatibility with current use case modeling techniques (e.g., SGAM, IEC 62559) should be assured. Furthermore, specifications for different parts will be needed (e.g., functionality, ICT specifications, information flows). With a formal specification and design method it can be used as a starting point for an automated engineering approach.
R2	*Implementation* Priority high This is an obvious step in an engineering methodology. It covers the implementation of functionality (e.g., control algorithms, automation functions, data processing) as well as the creation of different configurations (e.g., ICT setups, information flow configurations). During the implementation of a function, the algorithm of the function is defined in a formal software specification (e.g., UML, IEC 61499).
R3	*Testing and Validation* Priority high Any implemented functionality and configurations should be tested and validated. This can be either through simulative or laboratory tests. With a simulative validation, a system is tested in a virtual environment. This is a common approach in cases where the real system cannot be used directly, either because it can cause damage (e.g., to persons, equipment, or cost) or because the system is not available. In a laboratory validation, the implemented application or parts of it are tested in a controlled environment, where the final system can be emulated. This is usually a power system laboratory where the necessary components are available to emulate the needed connections for the application (e.g., AC/DC connections, ICT connections).

R4 *Release and Deployment* Priority high
This requirement concerns the process of making the implemented application ready for use with a certain system (i.e., power system and ICT system). This means that any functions and configurations must be prepared for their host platform. To operate the implemented application, it must be deployed to the field. This includes installation of any new system hardware components, software functions, and configuration of the communication and ICT system. In order to reduce the down-time of the system, it is important that the deployment is as rapid as possible.

R5 *Seamlessness* Priority high
During the whole rapid engineering process the transition between one engineering step to another must be as seamless as possible. Consequently, the best possible transition is automatic and does not require any user input. Still the methodology is not allowed to take away the control from the user and it must always be possible for the user to interact with the process.

R6 *Rapidness and Effort* Priority high
Smart grid solutions are becoming more and more complex, which results in increasing engineering efforts and costs. Therefore, it is important to improve the rapidness of traditional engineering methods. With reduced development effort, the time between specification and deployment of smart grid applications is also reduced. This will hopefully also increase the acceptance for new smart grid solutions, since the investment risk decreases.

R7 *Correctness* Priority high
Due to the multidisciplinary character of smart grid applications, this also requires the engineer to have an expert knowledge in each discipline. This is often not the case, which increases the risk of human errors. Therefore, one requirement for the rapid engineering methodology is correctness of the implemented applications.

R8 *Domain Expertise* Priority medium
The rapid engineering methodology should allow experts from different domains (e.g., control, communication, power system) to participate in the modeling process. This means that notation and syntax should be kept, as far as possible, for each of these domains.

R9 *Handling Legacy Systems* Priority medium
Grid operators expect a long service life of all components in their systems. Since not all components are changed at the same time it must be possible to handle already existing legacy systems. This means that the rapid engineering methodology must be able to integrate these in terms of existing configuration and functions.

R10	*Interoperability*	Priority low
	Interoperability is a critical issue in smart grid applications. This must be assured on all levels, from specifications over implementation, to deployment and finally during operation. Also components from different manufacturers must be handled, which requires a manufacturer independent method.	
R11	*Changing Requirements*	Priority low
	The engineering process must be flexible enough to handle changing requirements on the developed application. Often functional as well as non functional requirements change during the development process. Therefore, at any time during the engineering process, it must be possible to consider new requirements.	

CHAPTER 4

Rapid Engineering Methodology

The usage of proper automation methods and corresponding tools can offer a huge optimization potential for the overall engineering process. A starting point for this improvement is detailed use case and requirements engineering. This results in a structured description of use cases. However, since this information is still in a non-formal representation, it cannot be adequately utilized in a computerized and automated approach. By collecting the use case information in a formal model this can be used for direct automatic code generation. But, up to now, there is no integrated approach available that covers the whole development process for smart grid automation applications using a MDE approach which can handle the multi-domain aspect of smart grids.

This work addresses these needs with a rapid engineering methodology. The goal with the methodology is to cover the overall development process—from use case design to deployment—for the development of ICT, control, and automation functions used in smart grid and power utility automation applications. Based on this goal it is clear that the methodology must cover a wide range of engineering activities. On one side, it should be possible to handle high-level use case design (e.g., according to SGAM or IEC 62559). On the other hand, it must also be possible to define functionality detailed enough that it can be executed on field devices. An overview of the rapid engineering process is illustrated in Figure 4.1.

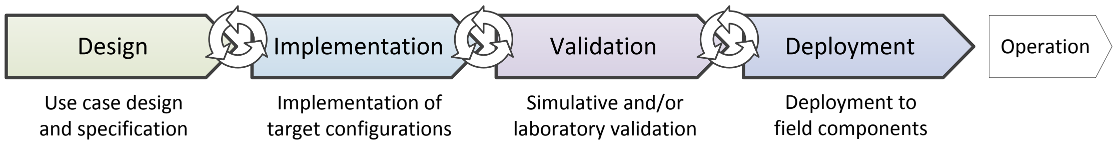

Design
Implementation
Validation
Deployment
Operation
Use case design and specification
Implementation of target configurations
Simulative and/or laboratory validation
Deployment to field components

Figure 4.1: Overview of the rapid engineering process.

The rapid engineering process in this thesis considers the four main development phases:

design, *implementation*, *validation*, and *deployment*. These phases are followed by the *operation* phase, where all activities related to the operation of the application are summarized. How the *operation* phase is carried out is not part of this thesis. The first four phases of the engineering process are encouraged by the research question in Section 1.2. Furthermore, they are also stated in the first four requirements in Table 3.2 (i.e., R1-Design and Specification, R2-Implementation, R3-Testing and Validation, and R4-Release and Deployment). These requirements are directly related to the different development phases of the engineering and deployment process.

The methodology presented in this work focuses on the design and implementation process of smart grid use cases. The intention is that the methodology should be usable for as many of the stakeholders in Table 3.1 as possible. However, since each stakeholder will also have their own special requests on the engineering method, it will not be possible to cover every single requirement within this work. In order to limit the work, the following non-goals are defined:

- It is not intended to use the methodology to model more than one use case at the time. If this should be supported, special considerations are needed regarding scalability and concurrency. Instead this is seen as future work.
- The methodology is, at the moment, not intended for use case requirements capture. It is assumed that a requirements engineering has been done before the design of the use case.
- It is not intended to use the method solely for the documentation of existing systems (e.g., a whole utility system with existing power, ICT, and automation infrastructure). Although it should be possible to model existing parts of a system it is the main purpose of the methodology to implement new functionality.
- The *operation* phase, as seen in Figure 4.1, is not considered in the developed engineering methodology. Instead the engineering process in this thesis ends with the deployment of the developed application.
- Although it would further automate the development process, automatic execution of *validation* phase in Figure 4.1 is not covered by the engineering methodology. The rapid engineering process provides automated support for the transitions to and from the *validation* phase.
- It is not a goal of this thesis to contribute to the research of software engineering methods. Instead, these methods are only used as tools in order to analyze the main research question of this work.

Different parts of the rapid engineering methodology have already been published in different publications. An initial concept as well as a discussion about the required modeling methods were shown in [5]. The main concept which is presented in this work was originally published in [117]. The rapid engineering methodology also contains aspects from other publications, especially [3, 9, 6, 8], and [116].

4.1 Rapid Engineering Concept

There are of course different paths to achieve the same goal. However, some reference points can be defined. Current description methodologies like SGAM and IEC 62559 represent a state-of-the-art for use case designs today. These approaches are also appropriate as starting point for the proposed rapid engineering methodology. At the end of the path executable and deployable code should be available for field automation devices. The concept for the rapid engineering approach was originally published in [117], and serves as a basis for the concept below.

Based on the main research question for this thesis, and the requirements in Table 3.2, the initial engineering approach, as seen in Figure 4.1, needs to be further defined. As already discussed in Section 1.2, software engineering concepts are introduced in order to automate the process from design to deployment. In Figure 4.2, a conceptual overview of the rapid engineering methodology is shown.

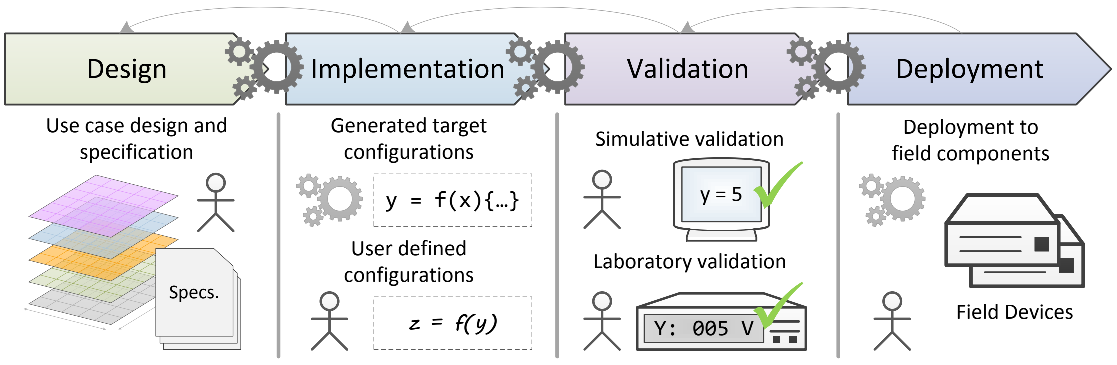


Figure 4.2: Conceptual view of the rapid engineering methodology.

The rapid engineering method should assist the user, and especially provide automated transitions between the different development phases. It should also support the user within the phases, for example, through automated generation of target configurations or automated deployment. Generally, it must also be possible to seamlessly move back and forth between the development phases. Subsequent changes in the *design* should not result in a complete redevelopment of the *implementation* phase. This can be compared with R11-Changing Requirements.

The development starts with the *design*. For this purpose, current use case description methods should be used as a reference. This is a phase that requires a lot of manual input. For example, for each layer in SGAM, specifications and designs are made by the user. This input is crucial since it makes up the main input for the *implementation* phase. In order to support an automated usage of the use case information, a formal and machine readable design method should be developed.

The second phase is the *implementation*. In this phase, target configurations are created based on the specification in the use case design. Two types of target configurations are in focus of this thesis: functions and communication configurations. The functions are algorithms or logic that will later be converted into executable code. Functions can either be created manually by the user or using an automated process (e.g., based on standards and other documented functions [66]). Based on the implemented functions, platform specific code is generated. In order to generate function code from the use case descriptions a formal implementation language must be provided. The communication configurations are used to configure the communication setup needed for the developed application. This includes configuration of the information sent between functions, but can also include low-level configuration of the communication network (e.g., assignment of the source and destination addresses).

In the *validation* phase, the generated target configurations are tested. For simulative validations, the target platform would be a simulation platform. In this case, the corresponding simulation model would be generated. A laboratory validation may also be a mixture of simulations and real system validations. The focus of this thesis is mainly to support the transition between the *implementation* and the *validation* phases. The actual validation is made by the user.

The last phase is the *deployment* of the generated, and validated, target configurations to field devices. This includes transferring the generated and compiled code to the devices and start the application. The goal is to provide the user with as much support as possible, and preferably as much of the deployment as possible should be automated.

In the following sections, the concept of the rapid engineering methodology is further elaborated. It starts with a discussion about needed models and modeling techniques. Then, different possibilities to use existing techniques for the different phases of the rapid engineering method are discussed. Thereafter, the development phases shown in Figure 4.2 are described in detail.

4.2 Needed Modeling and Design Methods

The concept for the rapid engineering methodology presented above should be further elaborated based on the requirements in Table 3.2. The general approach in this thesis is to try to reuse existing methods as far as possible, before inventing something new. There already exists a multitude of modeling, design, development, and engineering methodologies that are being used for other domains (e.g., software engineering, factory automation, automotive, aviation). Furthermore, as already mentioned in Section 1.2 models, and the extraction/insertion of information, are key components for the rapid engineering method. Therefore, this chapter discusses how existing methods and models from the smart grid domain can be applied to the different phases of the rapid engineering method. The discussion below is mainly based on [5, 8, 9].

As already discussed, the current state of practice provides a limited support for formal and structured specification and design of smart grid applications. Until now, the need for such descriptions was not very high since the number of complex applications was rather limited. However, with the introduction of advanced ICT-based automation systems this is changing. The electric energy system is moving towards a Cyber-Physical Energy System (CPES).

With the transition towards a CPES, power system engineers are confronted with multidisciplinary problems. In the past, it was enough for the engineer to be an expert in the power system domain. Today, the engineers must also be able to handle ICT, automation and control topics. In order to support the development of such applications, it is important to allow experts from different domains to participate in the engineering process, see requirement R8-Domain Expertise. To support this, the rapid engineering method in this thesis uses the concept of Domain Models (DMs). For each domain, the DM should allow a domain-specific engineering by experts in this domain. Finally, it is the goal with the rapid engineering methodology that the DMs can be combined into one holistic model.

This thesis focuses on DMs from the power system, communication, automation and control domains, as well as a corresponding use case design method. It is important that the DMs are as platform independent as possible. This to ensure that they are not limited to a certain platform. The following sections study already defined models and standards used for use case design and in the power and energy, ICT and automation domains. By ensuring that relevant standards and models are used, not only R8-Domain Expertise, but also requirement R10-Interoperability can be largely satisfied. Furthermore, if the DMs are chosen such that they are compatible with existing system standards and models it is also possible to fulfill requirement R9-Handling Legacy Systems.

4.2.1 Specification and Use Case Design

For a larger system, especially a CPES, with multiple applications, a use case design language will be crucial in order to understand the correlation between different components, e.g., DERs or smart meters. Starting with use case description methods like SGAM and IEC 62559, a structured approach for use cases design can be defined. For example, a structured description of the actors and their interactions simplifies the development process. However, the proposed use case methodology based on the SGAM has no standardized way of representing the objects, e.g., controllers and power grid components, neither in the way they are depicted, nor in the semantics used in the description. In order to achieve a common description methodology, the depiction as well as the semantics of such objects need to be standardized. Furthermore, to fully take advantage of the high amount of information in the modeled use cases, a formal machine-readable format needs to be used.

In this work, SGAM, IEC 62559, and current formal specification approaches [31, 5, 116] are used as a basis to define a formal specification and use case design method. Apart from use case design, the methodology must also support a number of different specifications with different levels of detail. This can for example be IEC 61850 SCL specifications that are used as inputs for the engineering. The use case design and the specifications must be combined into a holistic use case design language.

From a user perspective, the specification and design is a phase of the engineering process that contains a lot of manual work. Therefore it is important that the design methodology helps the user. For this purpose, it is also important that the final design methodology can be integrated into different software tools. Using this approach it should be possible to fulfill requirement R1-Design and Specification. Also, using state-of-the-art design methods (i.e., SGAM, IEC 62559), it is ensured that a domain-specific format, suitable for smart grids, is used.

4.2.2 Power System Domain

The power system DM should be a representation of the power and energy grid itself, with components (e.g., breakers, lines, transformers, loads, generators) and their dynamics as well as the topology of the grid (i.e., how the components are connected with each other). With CIM, a great attempt has already been made to standardize modeling and description of transmission systems as well as distribution systems. It contains different models for multiple grid components. Some of these components are shown in Figure 4.3 where an example network is modeled using CIM. Since CIM is described using UML the names in the boxes represent CIM classes (e.g., *PowerTransformer*, *Breaker*, *SynchronousMachine*).

An advantage with CIM is that it is supported by many power system simulation tools [112]. This means that it is possible to use already existing models from these tools and import them into the rapid engineering approach. The opposite would also be possible, which allows a fast deployment of power system models into simulation tools.

In this work, CIM will be used as a basis for the power system DM. The main goal with the power system DM is to provide the user with a possibility to model the static parts of the power system. Thus it is not in focus to allow the modeling of component dynamics.

4.2.3 Communication and ICT Domain

The communication and ICT DM should be able to describe the communication network used in the power system as well as provide an information model for the data exchanged in the network. It is also important to take into account the huge amount of communication protocols and standards already in use in the power and energy system today. Therefore it must also be possible to model commonly used communication protocols (e.g., PLC, Ethernet, Industrial Ethernet, Transmission Control Protocol (TCP), User Datagram Protocol (UDP)), but also existing information models (e.g., CIM, IEC 61850). The

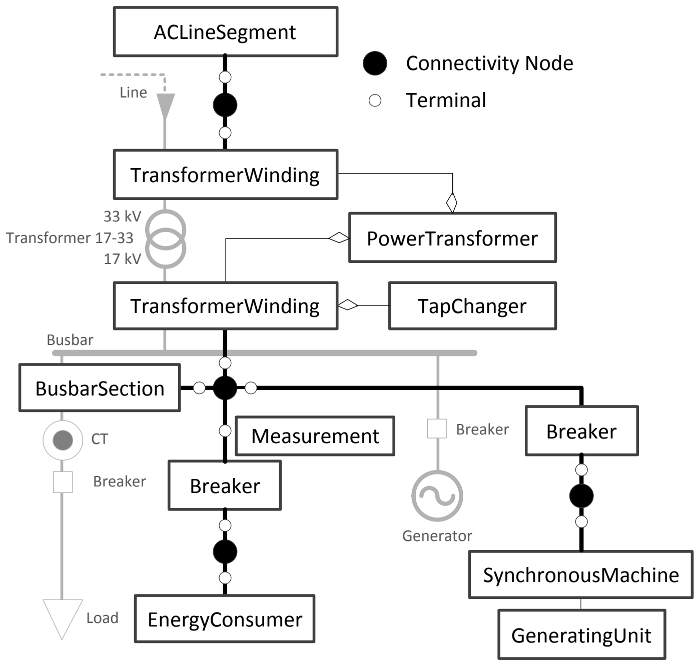


Figure 4.3: Simplified representation of the power system DM based on CIM

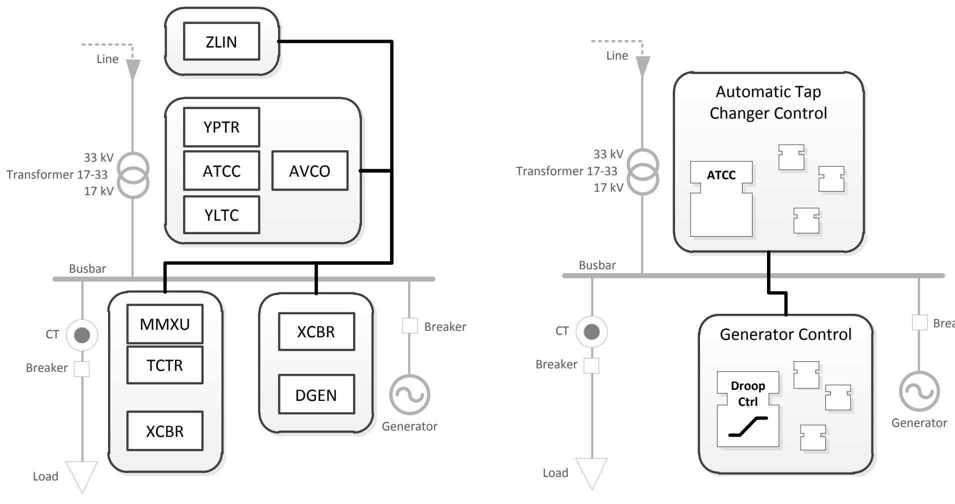
communication standard IEC 61850 already has an information model designed to fit power utility applications.

Especially the object-oriented information model of IEC 61850 provides a good starting point for a communication DM. IEC 61850 uses LNs to model network components as well as functionality provided by the components (e.g., *YPTR* is the LN for a power transformer and *ATCC* is the LN for an automatic tap change controller). In Figure 4.4a the communication DM, modeling the same exemplary network as in Figure 4.3, is shown using IEC 61850 LNs.

IEC 61850 was created especially with power utility automation applications in mind. But, there are many other cases of data exchange where IEC 61850 may not be the best choice, or may not be possible to use. Furthermore, sometimes a more generic information model may be needed. For example, during the use case design it is not always possible to make an unambiguous definition of the data that is to be exchanged. However, this is a well known problem in computer science, where IDLs (e.g., [74, 133]) are used to described the information in a platform-independent way. In a second step, the interface description using the IDL can be generated into specific code, for example Java or C++ (see also Section 2.5.2).

For the rapid engineering methodology, a communication DM is needed that can handle both smart grid specific data models as well as generic information descriptions. Furthermore, the communication DM must also be able to model different types of services.

Common patterns are client/server or publish/subscriber based communication.



(a) (b)

Figure 4.4: Simplified representation of communication and automation DMs: (a) Communication DM based on IEC 61850; (b) Automation DM based on IEC 61499

4.2.4 Automation and Control Domain

The automation and control DM should address the different control and automation functionalities of the smart grid. Two examples are discussed in BC2, where a coordinated voltage controller is developed, and in BC3, where the implementation of ancillary services for DERs is shown. It must also be possible to model the interactions between different control functions (e.g., the coordinated voltage controller sends voltage set points to the local DER controller). In cases where two interacting functions are executed on different hardware devices, communication is needed. In that sense, the automation DM and the communication DM are very much related with each other.

For the definition of the automation DM, existing methods and standards should be considered. As mentioned in Section 2.3, many automation approaches and concepts, as well as corresponding international standards, have already been developed supporting the implementation of automation applications (e.g., IEC 61131-3, IEC 61499). However, non of these standards were specifically developed for smart grid applications. A closer look at these concepts shows that a number of points are still open and need to be addressed: *(i)* formal design of automation functions on a platform-independent level would improve the interoperability and portability of developed control solutions; *(ii)* the exchange of automation models between different engineering tools, and between different stakeholders, is still an open and only partly solved issue; *(iii)* despite the already available domain standards for power and energy systems (e.g., CIM, IEC 61850) as well as for automation approaches (e.g., IEC 61131-3, IEC 61499), a harmonization and integration of the different concepts is necessary in order to achieve a comprehensive automation model for smart grid systems.

There are multiple description languages for architecture design of automation systems in other domains (e.g., UML, SysML, AADL) [148]. System and architecture design for automation concepts in smart grids can also partly be found in CIM or IEC 61850. However, even if these two standards can be used to define a system architecture, they do not provide any means to formally specify automation functions and algorithms.

Several standards and formal design approaches have also emerged from the automation industry. Two of the most established standards are IEC 61131-3 and IEC 61499. Several papers have already compared these two standards to be used for automation functions in smart grids [60, 146, 149, 170]. Summarizing, IEC 61499 has two main advantages compared to IEC 61131-3 as automation DM for smart grids. First of all IEC 61499 was designed for modeling of distributed automation applications, which is highly the case with power utility automation. Secondly, IEC 61499 also includes a system model, where mappings between hardware and software functions can be made. Such a system model is not available in IEC 61131-3. In Figure 4.4b, a simplified example of the automation DM using IEC 61499 is shown.

4.2.5 Software Engineering Methods

In order to use the different DMs together they need to be combined into a holistic smart grid model. Since the DMs are based on different domain-specific models and standards the combination must be based on the semantic similarities between the DMs. When combining different DMs together, it is important to keep them synchronized. This is necessary, since with more sources of information, the same information is bound to exist in more than one model. This can cause conflicts if the sources are not synchronized. The models can be synchronized manually, but to minimize design and implementation errors, an automatic approach is preferred. By using automatic synchronization between the DMs it is possible to keep their information unambiguous (i.e., changes in one DM should lead to automatic changes in the other DMs and vice versa).

When comparing the different DMs in Figure 4.3 and in Figure 4.4, it is clear that there is a certain overlap. Some of the components are represented in all DMs, but each DM models a different functionality of the component. For example, for the *Generator* its generating capabilities are represented as a *SynchronousMachine* in Figure 4.3 and as a *DGEN* LN in Figure 4.4a. In Figure 4.4b, only the droop control functionality of the *Generator* is modeled.

One approach to synchronize the DMs is to use model-to-model transformations (often only called model transformations). Model transformations are an important part of MDE methods, see Section 2.5.1. In the MDA initiative, transformations are used to create a PSM from a PIM. The same concept can be used to synchronize the DMs. Model transformations enable a domain expert to make a system model in their own DM (e.g., the control expert makes a control model) and afterwards generate the other

DMs through transformations. For this task rules have to be specified that define which transformations are allowed between the different parts of the DMs.

If model transformations are automatic they are also a powerful tool to accelerate the engineering process. First of all they can be used to achieve seamless transitions between different engineering steps, see requirement R5-Seamlessness. Secondly, the more parts of the development process that are automated the more rapid is the engineering method, and the less human effort is needed, see R6-Rapidness and Effort. One specific example is generation of platform-specific code (i.e., model-to-code transformation). This can be code for simulation models, communication specifications, as well as field implementations used in operation. If model transformations are done in a correct way they are also a possibility to minimize the amount of human errors [128, 47]. Consequently, transformations also contribute to the realization of requirement R7-Correctness.

4.3 Phase I: Design and Specification

The *design* phase is the first phase of the rapid engineering methodology. Thus it is also the most important phase. It is the phase where the user provides the most input. This input is also needed in order to complete the following phases. Consequently, the higher the quality of the user design, the higher the possibility of a successful development. In order to increase the quality, an appropriate design support is needed. A structured approach and guidelines are two ways to support the user during the design phase.

Based on the SGAM use case approach, and as a response to requirement R1-Design and Specification, a structured design process is defined. It is depicted in Figure 4.5. As already shown in Section 2.2.2, the first step of the SGAM approach is the use case analysis. It is assumed that the user has done a general analysis before the design and specification starts. The analysis should contain a narrative summary of the use case and it should also contain a first requirement identification. Based on this use case analysis, the user makes a detailed description of the application in different specifications.

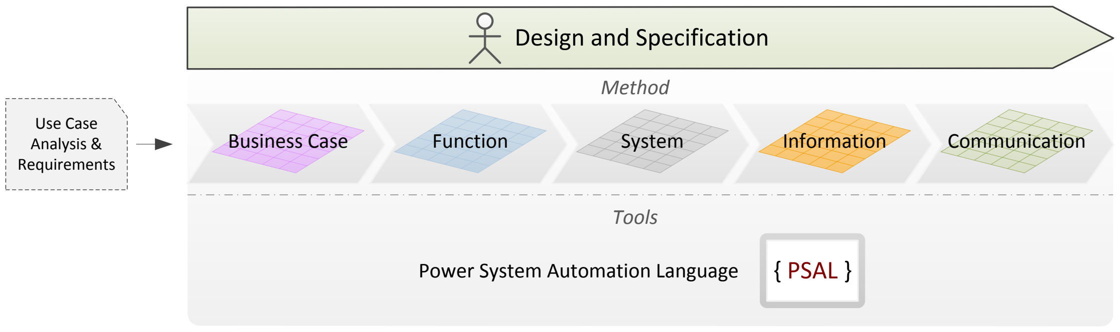


Figure 4.5: Overview of the *design* phase and its different steps.

In the *business case* specification, a general overview of the business cases and goals, which are of importance for the application, are specified. This is the first step of the *design* phase after the use case analysis, as seen in Figure 4.5. The engineering methodology in this thesis provides the user with the possibility to define a business case with a name and a narrative summary, as well as a description of the associated business goals. A business case can in turn be implemented by a number of sub use cases and functions.

Once a business case is defined this can be divided into functions. This is done in the *function* design. First of all functionalities of the application are specified. From this it should be clear what the goals of each function are, what inputs are needed, and what outputs are produced. The functional design should also include a description of how the functions interact with each other. Thus it is also necessary to make an initial specification of the information that is exchanged in the application. This is the second part of the *function* design.

Once the functional aspects of the application are defined it must be specified in which system they should be executed. Therefore, the next step is the *system* specification. This includes the description of the physical system which is used for the application. It can be power system equipment as well as ICT equipment. Furthermore, connections between the components are also specified. Finally, this step also includes a definition of a function-system mapping, an assignment of software to execution hardware.

Depending on the functionality of the application and the execution system, different information models may be suitable. Thus, the next step in the *design* phase is to make a detailed *information* specification. An initial description of the information is provided during the *function* design. In this step, this initial description is updated and if needed assigned to a specific data model.

The last part is the *communication* specification. Here, the user defines communication related parameters. This can for example be a specific communication protocol for a certain Open Systems Interconnection (OSI) layer. It can also be other parameters like Virtual Local Area Network (VLAN) identifiers, priority, or Media Access Control (MAC) and Internet Protocol (IP) addresses. With this last part of the *design* phase, the whole of requirement R1-Design and Specification is covered.

The design steps in Figure 4.5 are intended as guidelines for the user and it is not necessary to follow them in a strict order. Indeed, it is very much intended that users adapt these steps to fit their own needs. In order to allow this, a tool is needed that covers all the different design steps. At the same time, it must be flexible enough to accommodate specific user needs. For this purpose, the Power System Automation Language (PSAL) was created. The following sections introduce and specify PSAL and show how it is used for the *design* phase. PSAL was first presented in [116] and [117]. These publications serve as a foundation for the PSAL specification below.

4.3.1 Power System Automation Language

The main intention with PSAL is to provide a formalized DSL for SGAM compatible use case design. It is the main tool for the *design* phase, as seen in Figure 4.5. This section outlines the parts of PSAL and how it relates to the SGAM approach. In the next section, the grammar and syntax of PSAL are formally defined.

Figure 4.6 shows the SGAM structure together with a simplified model of PSAL represented as a UML class diagram. The different SGAM layers also correspond to the different design parts in Figure 4.5 and therefore this structure must be represented in PSAL as well. The *system* specification is used to describe the hardware components of the system. Generally, these are either power system or ICT components. In PSAL, a general `Component` is provided. Specializations thereof are used to represent power system and ICT components. `Components` either already exist in the system or they represent new equipment. The `Device` represents an ICT controller hardware, where algorithms or software can be executed in so-called `Resources`. In order to connect `Components` with each other, they can provide one or more `PhysicalInterfaces` (i.e., power or ICT interfaces). Two `PhysicalInterfaces` can be connected with each other through a `Connection`.

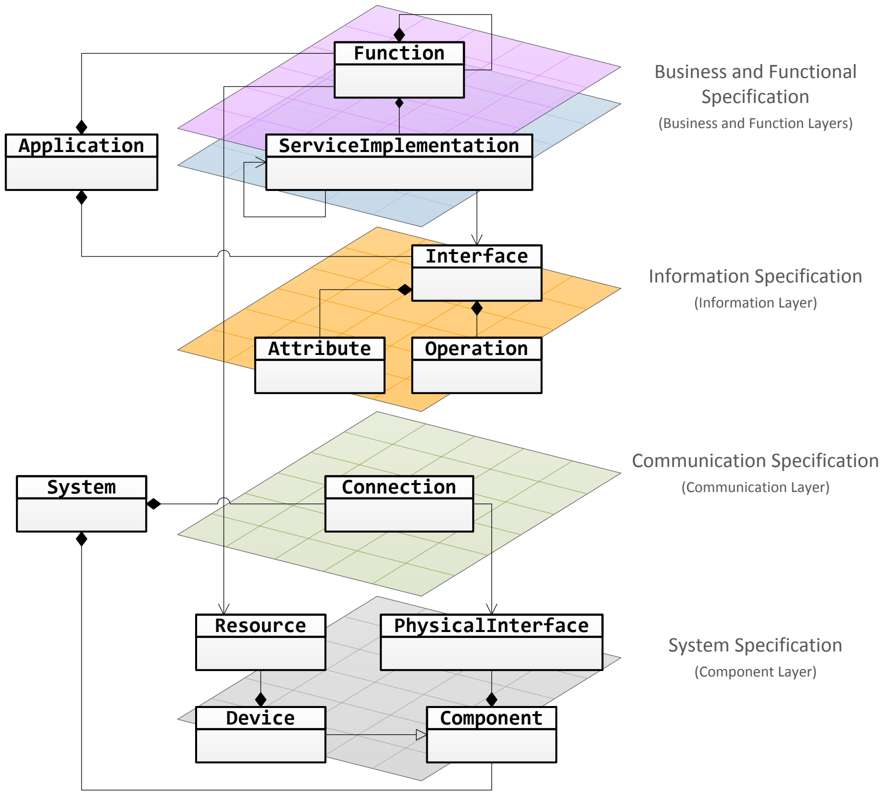


Figure 4.6: Simplified UML representation of PSAL on top of the SGAM structure.

If a `Connection` connects two ICT interfaces this is defined as an ICT `Connection`. These two parts are also used for the *communication* specification. As described above, this part is used to define different communication parameters (e.g., IP or MAC addresses).

In PSAL, these specifications can be modeled within the ICT `Connections` and the `PhysicalInterfaces`.

Although the communication protocols are specified in the *communication* specification, the information model that is used is specified in the *information* specification. With PSAL, a detailed specification of the data exchange is realizable. According to requirement R1-Design and Specification it should be possible to describe the information exchange between two parties. In particular, this means two things: it must be possible to specify the format of the exchanged data (i.e., what information model is used) and it must be possible to specify how the information is exchanged (e.g., if it is published or requested). PSAL provides two main objects to model the information exchange: `Events` and `Interfaces`. The `Interface`, as seen in Figure 4.6, models a client/server pattern and allows data to be exchanged either through read and write of `Attributes` or by calling `Operations`. Similarly, `Events` can also be defined to model a publish/subscribe pattern that allows data to be published in one direction.

The *business case* specification is used to describe business cases and goals. Each business case can in turn be divided into several use cases. This process continues in the *function* design, where use cases and sub use cases are divided into functions. In order to model this hierarchy, PSAL provides `Functions`. Each `Function` can contain other `Functions`, which makes it possible to model the same hierarchical structure as with business cases, use cases, and functions.

Different approaches can be used to model functions and their interaction with each other. The SGAM methodology proposes to describe functions in different UML diagrams (e.g., use case, activity, and sequence diagrams [25]). In order to model function interactions in PSAL, `Functions` can have `ServiceImplementations`. Each `ServiceImplementation` implements an information interface (e.g., an `Interface`). Thus by connecting `ServiceImplementations` with each other, it is also possible to model the information flow between the `Functions`.

Additionally to the SGAM methodology, PSAL provides the `Application` and the `System` model parts. Together they form a further abstraction layer. The `System` contains the specification of the component and the communication layers. The `Application` contains the design of the business, function, and the information layers. In order to associate the `Application` model with a `System` model, `Functions` are mapped to `Resources`. A `Function-Resource` mapping defines in which `Resource` a specific `Function` is executed.

In SGAM, the placement of component and functions into domains and zones was introduced. Although this placement helps to structure a use case, it is mainly for documentation purposes and has no direct influence on the realization. With a textual implementation of PSAL, the definition of domain and zone through spatial layout is not an option. Nevertheless, it can be done through special comments or by using Java-like annotations (e.g., `@DER` and `@Station`) in the source code.

4.3.2 PSAL Specification

Based on the considerations in the previous section, a syntax for PSAL is defined using Extended Backus–Naur Form (EBNF) [33]. A complete overview of the grammar can be found in the Appendix A. As depicted in Figure 4.6, the `System` and the `Application` are two main parts. Apart from these, `Module`s can also be directly, in order to simplify the definition of communication protocols and information models. The main parts of the PSAL source code document are shown in Listing 4.1.

```
PSAL ::= PsalContent*
PsalContent ::= System | Application | Module
```

Listing 4.1: Main parts of a PSAL source document.

This means that a `PSAL` document can contain zero or more occurrences of `PsalContent`, which is either a `System`, an `Application`, or a `Module`. In order to define the domain and zone, a general annotation syntax is defined in Listing 4.2. The annotations can be placed before any `Function` or `Component` in order to place this part in a certain domain and zone.

```
Annotation ::= '@' ID AnnotationParams?
AnnotationParams ::= '(' AnnotationParam (',' AnnotationParam)* ')' | '(' ')'
AnnotationParam ::= ID
```

Listing 4.2: Annotations in PSAL.

There a number of objects in PSAL that require the user to specify different parameters. Often these parameters are platform specific and thus it is not possible completely define all of these parameters in the PSAL grammar. Instead a generic syntax is defined that can be used for parameter assignments. It is always a pair with an identifier and a string value, as is shown in Listing 4.3.

```
IDValuePairBody ::= '{' IDValuePair* '}'
IDValuePair ::= ID '=' ValueLiteral
```

Listing 4.3: Parameter definitions in PSAL.

Comments are also allowed and should be used for documentation. Multi-line comments are started with `/*` and ended with `*/`. These comments can span multiple lines but do not nest. A single-line comment is started with `//` and terminates at the end of the line. Principally comments may contain any characters that do not terminate the comment.

System Model

The `System` model is used for the *system* and the *communication* specification. The `System` is defined in Listing 4.4. Each `System` has a unique identifier and can contain zero or more `SystemComponents`. These are either a `Component` (see Listing 4.5) or a `Connection` (see Listing 4.10).

```
System ::= 'system' ID '{' SystemContent* '}'
SystemContent ::= Component | Connection
```

Listing 4.4: The `System` model.

For this work, two different types of `Components` can be defined, as is seen in Listing 4.5. These are either `ICTComponents` or `ElectricalComponents`.

```
Component ::= ICTComponent | ElectricalComponent
```

Listing 4.5: The `Component` model.

This work focuses on programmable `ICTComponents`, which are represented by `Devices`, as seen in Listing 4.6. A `Device` has a unique identifier and can contain `Resources` and `PhysicalInterfaces`. Other ICT components can also be defined. At the moment `gateways`, `switch`es and `routers` can be defined in order to represent an ICT infrastructure.

```
ICTComponent ::= Device | OtherICTComponent
Device ::= 'device' ID '{' DeviceContent* '}'
DeviceContent ::= Resource | ICTInterface
Resource ::= 'resource' ID
OtherICTComponent ::= OtherICTComponentTypes ID IDValuePairBody?
OtherICTComponentTypes ::= 'gateway' | 'switch' | 'router'
```

Listing 4.6: The `ICTComponent` model.

In order to connect `Devices` with each other, `ICTInterfaces` are used. They can be of different types in order to represent different ICT technologies. Listing 4.7 shows these types. The `ICTInterface` can also contain `IDValuePairs` in order to define parameters like an IP address or a port (see Appendix A).

```
ICTInterface ::= ICTInterfaceType ID IDValuePairBody?
ICTInterfaceType ::= 'ethernet' | 'wireless' | 'serial' | 'analogue' | 'digital'
```

Listing 4.7: The `ICTInterface` model.

Apart from the `ICTComponent` it is also possible to specify `ElectricalComponents`. These are used to define the electrical power system upon which the application will operate. In this sense, the `ElectricalComponent` model is the representation of the power system DM in PSAL (see Section 4.2.2). As already discussed, there are multiple existing models available for representation of power system topologies. For the purpose of the rapid engineering methodology in this work, CIM is used as a reference [68]. Figure 4.7 shows an overview of the CIM objects that were used in PSAL and Listing 4.8 shows the `ElectricalComponent` model.

```
ElectricalComponent ::= Transformer | ElectricalEquipment
Transformer ::= 'transformer' ID '{' TransformerWinding+ '}'
TransformerWinding ::= 'winding' ID '{' ElectricalEquipmentContent* '}'
ElectricalEquipment ::= ElectricalEquipmentType ID '{' ElectricalEquipmentContent* '}'
ElectricalEquipmentType ::= 'generator' | 'line' | 'eswitch' | 'consumer' | 'busbar'
```

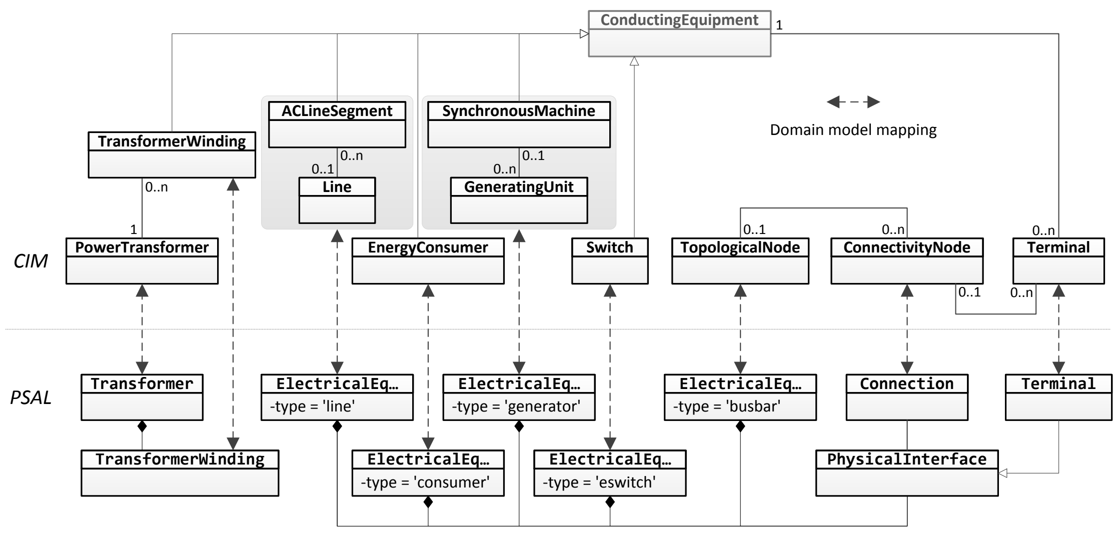


Figure 4.7: Relationship between CIM and PSAL.

```
ElectricalEquipmentContent ::= Resource | PhysicalInterface | IDValuePair
```

Listing 4.8: The `ElectricalComponent` model.

As is seen in Figure 4.7 and in Listing 4.8, the `ElectricalEquipment` can have multiple computing `Resources`, as well as multiple `PhysicalInterfaces`. This can both be `ICTInterfaces` and `Terminals`. The `Terminal` is defined in Listing 4.9 and is used to connect power system components with each other. Since `ElectricalEquipment` can contain both `ICTInterfaces` and `Terminals` it is also possible connect them with `ICTComponents`. Furthermore, all `PhysicalInterfaces` can also contain `IDValuePairs` in order to define different settings.

```
PhysicalInterface ::= ICTInterface | Terminal
Terminal ::= 'terminal' ID IDValuePairBody?
```

Listing 4.9: The `PhysicalInterface` model.

As shown in Figure 4.6, `Connections` can be defined between the `PhysicalInterfaces` of the `Components`. To define a `Connection`, the definition in Listing 4.10 is used. Generally, the `QualifiedNames` must match `PhysicalInterfaces` defined in the scope of the `System`. However, `Connections` directly to/from `OtherICTComponents` are also allowed. Optionally, a `Connection` can also be configured with user defined settings (i.e., `IDValuePair`). This can, for example, be communication network settings.

```
Connection ::= 'connect' QualifiedName 'with' QualifiedName IDValuePairBody?
```

Listing 4.10: The `Connection` model.

Application Model

The `Application` model in Figure 4.6 contains two main parts: `Functions` and information `Interfaces`. Furthermore, `Connections` between `ServiceImplementations` can be used to model the information exchange between `Functions`. The `Functions` and the `Connections` are used for the *business* and the *function* design, and the `Interfaces` are used for the *information* specification. The `Application` model is defined in Listing 4.11.

```
Application ::= 'application' ID '{' ApplicationContent* '}'
ApplicationContent ::= Function | Connection | Module
```

Listing 4.11: The `Application` model.

The main idea with the `Interface` in Figure 4.6 is to model the information that is exchanged between the `Functions` (i.e., the *information* specification). However, as already discussed in Section 4.2.3 this is a well known problem in computer science. There exist several IDLs to described a platform-independent information exchange [74, 133]. After comparing the syntax and features of different IDLs, the OMG IDL [74] was chosen as the most suitable for PSAL. OMG IDL is completely platform independent, and it uses a syntax very similar to the rest of PSAL. Figure 4.8 shows the relationship between the main objects of the OMG IDL and PSAL are shown.

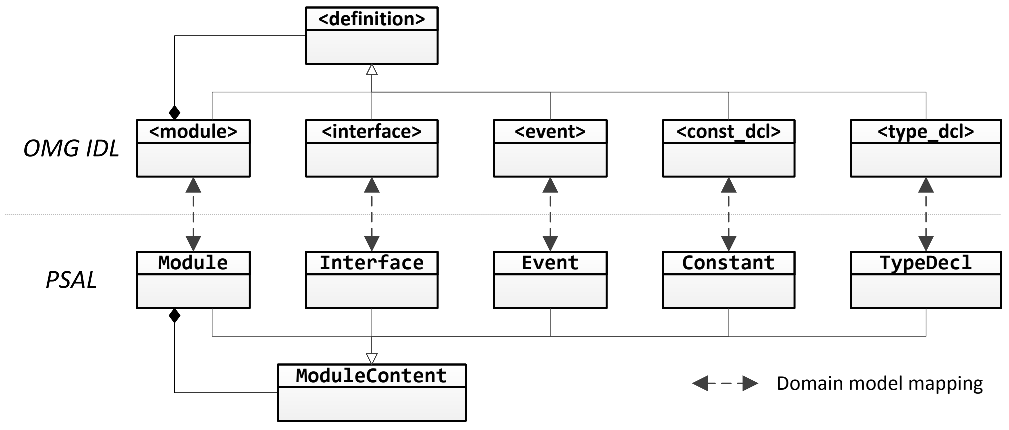


Figure 4.8: Overview of the main object mapping between OMG IDL and PSAL.

The `Module` is the top-level object that is used for the *information* specification. It is also the main part of the communication and ICT DM, as discussed in Section 4.2.3. The main function of the `Module` is to act as a container for other objects. It can also contain other `Modules`. The definition is shown in Listing 4.12.

```
Module ::= 'module' ID '{' ModuleContent+ '}'
ModuleContent ::= Module | Interface | Event | Constant | TypeDecl
```

Listing 4.12: The `Module` model.

For PSAL, the main objects for information exchange are the `Interface` and the `Event` elements. These also have a representation in OMG IDL and the mapping for the interfaces and the events are shown in Figure 4.9.

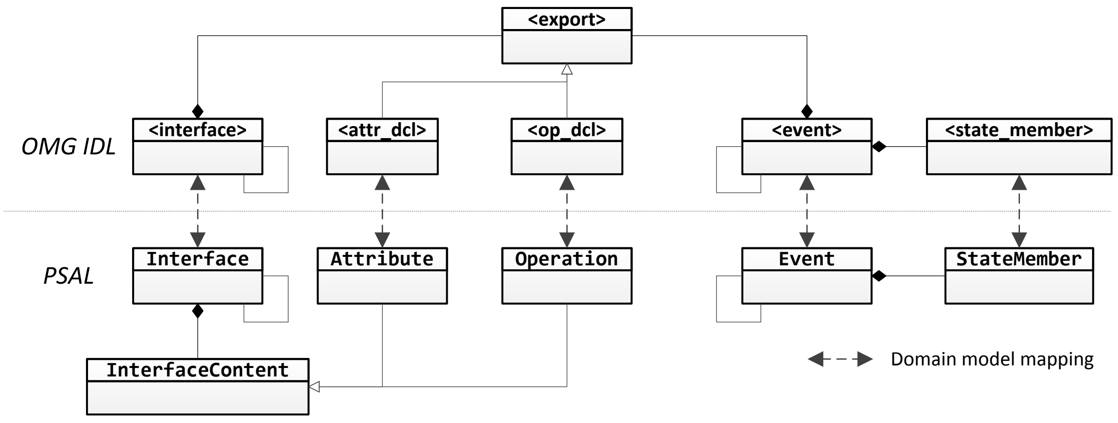


Figure 4.9: Overview of the mapping between the interfaces and events.

The `Interface` is intended for client/server based information exchange. A server provides an `Interface`, which is used by a client. Basically information can be exchanged either through `Attributes` or `Operations`. The `Attributes` are used for simple read and write operations. The `Operations` are services that are called by the client. They can have input as well as output parameters. Furthermore, inheritance of another `Interface` is also possible. As a result, the child has access to all the `Attributes` and `Operations` of the inherited parent. The `Interface` model is defined in Listing 4.13.

```
Interface ::= ('abstract')? 'interface' ID InterfaceInheritance? '{' ⤶
    ↳ InterfaceContent* '}'
InterfaceInheritance ::= ':' QualifiedName
InterfaceContent ::= Attribute | Operation | Constant | TypeDecl | ⤶
    ↳ CommunicationParameter
Attribute ::= ('readonly')? 'attribute' TypeSpecification ID (',' ID)*
Operation ::= ('oneway')? OpTypeSpecification ID OpParameters
OpTypeSpecification ::= TypeSpecification | 'void'
OpParameters ::= '(' Parameter (',' Parameter)* ')' | '(' ')'
Parameter ::= ParamAttribute TypeSpecification ID
ParamAttribute ::= 'in' | 'out' | 'inout'
```

Listing 4.13: The `Interface` model.

The second main object for information exchange is the `Event`. Unlike the `Interface`, it is used for publish/subscriber communication. A publisher emits an `Event` and the the subscriber consumes the `Event`. In PSAL, the information emitted by `Events` is represented using `StateMembers`. As with the `Interface`, `Events` can inherit from each other. The `Event` model is defined in Listing 4.14. For a complete definition of the *information* specification refer to Appendix A.

```
Event ::= ('abstract')? 'eventtype' ID EventInheritance? '{' EventContent* '}'
```

```
EventInheritance ::= ':' QualifiedName
EventContent ::= TypeDecl | StateMember | CommunicationParameter
StateMember ::= ('public' | 'private') TypeSpecification ID
```

Listing 4.14: The `Event` model.

The information defined using the `Interfaces` and the `Events` can be used by the `Functions`. In Listing 4.15, the `Function` model is defined. Each `Function` has a unique identification, a possible mapping (see Listing 4.17), and it consists of other `Functions`, `Connections` and/or `ServiceImplementations`. The `Function` is the main object used for the *function* design. It is also an important basis for the automation DM presented in Section 4.2.4.

```
Function ::= 'function' ID (FunctionMapping)? '{' FunctionContent* '}'
FunctionContent ::= Function | Connection | ServiceImplementation
```

Listing 4.15: The `Function` model.

A `ServiceImplementation` implements a service type—an `Interface` or an `Event`. `ServiceImplementations` can be requested or provided by a `Function`. Depending on the service type the syntax is a bit different, as seen in Listing 4.16. To define a service type, the `QualifiedName` of a `ServiceImplementation` must match a defined `Interface` or `Event`. `ServiceImplementations` can also override the communication protocol defined by an `Interface` or `Event`. This is done with the `ProtocolUsage`, where the `QualifiedName` must match an `Interface` or `Event`.

```
ServiceImplementation ::= ProvidedService | RequestedService
ProvidedService ::= ProvidedIDLInterface | ProvidedIDLEvent
RequestedService ::= RequestedIDLInterface | RequestedIDLEvent
ProvidedIDLInterface ::= 'provides' QualifiedName ID ('{' ParameterAssignment* '}')? ⤶
    ↳ ProtocolUsage?
ParameterAssignment ::= QualifiedName '=' ValueLiteral
RequestedIDLInterface ::= 'requests' QualifiedName ID ('{' ParameterAssignment* '}')?
ProvidedIDLEvent ::= 'emits' QualifiedName ID ('{' ParameterAssignment* '}')? ⤶
    ↳ ProtocolUsage?
RequestedIDLEvent ::= 'consumes' QualifiedName ID ('{' ParameterAssignment* '}')?
ProtocolUsage ::= 'using' QualifiedName
```

Listing 4.16: The `ServiceImplementation` model.

As represented in Listing 4.11 and in Listing 4.15, both the `Application` and the `Function` can contain `Connections`. In both cases, the syntax is the same as defined in Listing 4.10. In the `Application`, a defined `Connection` connects a requested service `with` a provided service. The same applies to `Connections` defined in a `Function` but with the additional possibility to connect a provided/requested service of a contained `Function` with a provided/requested service of the containing `Function`. In all cases, the connected `ServiceImplementations` must be of the same service type.

To connect the `Application` with a `System`, the `Functions` of the `Application` must be assigned to the components defined in the `System`. This is done by mapping a `Function` to a `Resource` in a `Device` or in an `ElectricalEquipment`. When a `Function` is mapped to a `Resource`, all provided and requested services of this `Function` are also mapped to the `Resource`. Consequently, these services are also available on any `ICTInterface` of the component. Listing 4.17 shows how a mapping is defined. The `QualifiedName` of the `FunctionMapping` must match a defined `Resource`.

```
FunctionMapping ::= 'at' QualifiedName
```

Listing 4.17: The `FunctionMapping` model.

4.4 Integration of an Implementation Language

With PSAL, a tool is provided that supports all parts of the design process as defined in Figure 4.5. However, with the `Functions` in PSAL the user can only make a high-level design of functions. Since one of the main goals of PSAL is to facilitate automated implementation of functions, the high-level `Function` design will not be enough. One approach could be to only offer a set of predefined `Function` types that can be combined into an application. This approach offers a clear simplicity, but also a lack of flexibility with respect to new or changing power utility automation functions. Another approach, which offers more flexibility, is to extend PSAL with a programming language that can be used to define `Functions`. However, since there already exist multiple programming techniques it is not feasible to define a new language for PSAL. Instead, PSAL is extended with an already existing implementation language.

Once an implementation language has been chosen, it must be defined how the language is used to implement the `Functions` specified by PSAL. Furthermore, it must also be defined how the `ServiceImplementations` can be implemented and how `FunctionMappings` are realized. Summarizing, the integration of an implementation language can be divided into the following guidelines:

1. Choosing an implementation language $\mathcal{L}$.
2. Define how $\mathcal{L}$ is used to implement `Functions`.
3. Define how `ServiceImplementations` are implemented using $\mathcal{L}$.
4. Define how `Connections` between `ServiceImplementations` are implemented.
5. Define how `FunctionMappings` are realized.

In this work, IEC 61499 is used as an implementation language. The following sections discuss how and why IEC 61499 was selected, how IEC 61499 is used for the implementation, and how `ServiceImplementations` and `FunctionMappings` are realized. The selection of IEC 61499 is mainly motivated by the discussions in [3, 6, 8], and [117].

4.4.1 Selecting an Implementation Language

Based on the definition of PSAL the following main concepts should either be directly supported, or implementable, by the language:

- *Software components*: The `Functions` in PSAL are simple software components and should be supported by the implementation language. Components should also be able to contain other components, in order to model different levels of detail.
- *Service implementations*: The software components must be able to implement and request services and this must be supported by the implementation language. It is also important that the language provides a support for connecting services with each other.
- *Execution platform mapping*: Once a software component is implemented it should be mappable to an execution platform. This mapping identifies on which hardware the software is executed.

Apart from the interoperability with PSAL, platform requirements may also need to be considered. Depending on the platform hardware and/or software some programming techniques may not be suited. For example, a system only consisting of embedded controllers with limited memory capabilities may exclude many higher order programming techniques. On the other hand, if a certain programming techniques has already been used for implementations in the system, this technique may probably be the best choice for compatibility reasons.

For this work, IEC 61499 was chosen as programming technique. IEC 61499 has already been discussed as one possible solution for implementing standardized smart grid applications [171, 147]. Especially, together with smart grid communication standards like IEC 61850 it has already been proven as a way of maintaining a consistent information model throughout the design for smart grids applications [60, 175]. Moreover, the application modeling capabilities of IEC 61499 can be exploited for an overall representation of whole automation application. Thus not only the single `Functions` can be implemented but also how the `Functions` are interacting in order to achieve a common goal.

IEC 61499 Application Modeling Approach

The IEC 61499 approach is centered around application modeling. First of all, the overall automation application is modeled and designed using FBs. This is done independently of where the FBs will be executed. Afterwards in a second step, the FBs of the application are mapped to their corresponding execution platforms (i.e., control devices). This also means that the application model can be seen as platform independent—independent of the type of device it is running. This can be utilized for the *implementation* phase.

IEC 61499 also offers a model element called Subapplication. It can be used to nest FB networks. This can be used to build hierarchical models and therefore to structure complex control applications. Furthermore, this matches very well with how the `Functions` are modeled in PSAL. Consequently, Subapplications are one possibility to implement the specified `Functions`.

The standard data connections between FBs and between Subapplications are used to transfer data between components. These can also represent communication connections between different components in the grid. In PSAL, the `ServiceImplementations` are used to specify the information interfaces of a `Function`. In IEC 61499, the Adapter modeling object can be used to describe such interfaces. To account for both provided interfaces and required interfaces an Adapter is divided into a socket part and a plug part. An Adapter definition summarizes multiple events and data connections into one connection element. This has a number of advantages: *(i)* the interface of the FB is formally defined, since a plug can only be connected to a socket if they are of the same Adapter type, *(ii)* an Adapter may define a bidirectional connection, and *(iii)* the total number of connection elements in the FB network is minimized [57].

Before it can be executed, the control application has to be assigned to an execution platform. This is done by mapping Subapplications and FBs to Resources contained in hardware Devices. Once the FBs in the control application have been mapped, they can be deployed. This also means that connections between FBs of different devices must be replaced with the communication services as defined by the `Interfaces` or `Events`. Moreover, these communication services can also be encapsulated into special IEC 61499 communication blocks, so called SIFBs. This is described in more detail in Section 4.4.2.

Considering these aspects of IEC 61499, it is clear that it fulfills the requirements needed by an implementation language. Software components can be represented using FBs and Subapplications. Service implementations can be defined using Adapters in IEC 61499. Finally, IEC 61499 also inherently supports the mapping of FBs to hardware devices. As also discussed in Section 4.2.4, IEC 61499 has many advantages compared to other solutions. It is also completely platform independent [177], which makes it usable to many of the identified actors in Table 3.1 and it increases the chances of fulfilling requirement R10-Interoperability. The selection of IEC 61499 as implementation language also completes the definition of the automation DM.

Implementation Guidelines

In order to support the user during the *implementation* phase, guidelines and design patterns have been specified. These guidelines should help power system engineers to design standard-compliant and interoperable control applications using IEC 61499. The guidelines are divided into two categories: one concerning the modeling of the control application, and the other concerning the deployment of the control application to the field after it has been designed.

Following an MDE approach, the model of the application should contain no platform specific information. The platform specific information is added after it has been mapped to its execution platform. Based on [178], the following guidelines formalize the steps of modeling a smart grid automation application:

- Top-down as well as bottom-up modeling can be applied with the proposed modeling approach if necessary/suitable for a specific control problem.
- Hierarchical control applications should be modeled using Subapplications on different layers. Consequently, the top-most layer should provide an overview of the different business cases. Below the different `Functions` from the *function* design are implemented, also using Subapplications.
- The Subapplication representing a `Function` should implement its corresponding behavior. Furthermore, the behavior of a `Function` is also indicated by its provided and requested `ServiceImplementations`. Following the hierarchical approach, another layer can be added where the implementation of each `ServiceImplementation` is contained within a Subapplication.
- The interface of a `Function` is defined not only by the `ServiceImplementations` it provides, but also by the `ServiceImplementations` it requires from other `Functions`. To clearly define these `ServiceImplementations`, Adapters should be used. If a `Function` is providing a `ServiceImplementation` this should be represented by an Adapter Socket and required `ServiceImplementations` are implemented as Plugs.

Following the guidelines above, a structured and platform-independent application is created. The next step is to transform this into a platform-specific implementation and finally deploy it to the field. For this process, the following guidelines are suggested [7]:

- Each Subapplication representing a `Function` is mapped to its corresponding Resource and device. Subapplications representing proprietary or already implemented `Functions` are not mapped.
- Platform-specific information is added to the Resource. This include SIFBs for communication and process interfaces, as well as configuration of platform specific parameters.
- Adapter connections used in the application to model the information flow between devices are replaced with SIFBs implementing the needed communication services. These SIFBs are inserted into the Resource where the Subapplication is mapped. This process can be automated by the used modeling tool.
- The actual deployment of the application may require specific configuration steps. This is depending on the used implementation environment and is covered in Section 5.

The following section shows how the IEC 61499 is mapped to PSAL. In order to help the reader to distinguish between elements from the different models they are differently emphasized in the following sections. Any model elements from PSAL are emphasized with a teletype font (e.g., `Function`). Model elements from IEC 61499 are emphasized with a sans-serif font (e.g., FB^{499}).

4.4.2 Mapping IEC 61499 to PSAL

Once an implementation language has been selected it must be mapped to the `Function` model of PSAL. This means that one or more entities of the language are used to represent the software component, represented by the `Function` in PSAL. If possible a one-to-one mapping is to prefer since this also allows for a simple round-trip transformation.

The guidelines presented above in Section 4.4.1 can be directly used to define a mapping between PSAL and IEC 61499, as seen in Figure 4.10. Following these, each `Function` of PSAL is mapped to a SubApp^{499} (i.e., Subapplication) in IEC 61499. The SubApp^{499} is defined by a SubAppType^{499}. A SubAppType^{499} can also contain other SubApps^{499}, which makes it possible to model multiple levels of `Functions` using IEC 61499. Furthermore, each `ServiceImplementation` is mapped to an interface of the SubAppType^{499}. `ProvidedServices` are mapped to Sockets^{499} and `RequestedServices` are mapped to Plugs^{499}. The mapping between `Functions` and SubApps^{499} is shown in Figure 4.10a.

The overall mapping between PSAL and IEC 61499 is shown Figure 4.10b. Due to the similarity between the models, the mapping is straightforward. The `Application` is mapped to the Application^{499} in IEC 61499. Since IEC 61499 also has a system model it is possible to find a model mapping for the `Devices` and `Resources` of PSAL. As already mentioned, IEC 61499 supports mapping of FBs^{499}/SubApps^{499} to Resources^{499}. Using this opportunity, the `FunctionMapping` of PSAL can be directly represented as a Mapping^{499} in IEC 61499.

With this mapping between PSAL and IEC 61499, two important parts of the rapid engineering methodology introduced in Section 4.1 can be created. The SubAppType^{499} is used by the engineer to implement `Functions` in detail. This allows the realization of the *implementation* phase in Figure 4.2, and it also allows the realization of requirement R2-Implementation. Secondly, the automated mapping and deployment provided by IEC 61499 can also be used for the *validation* and *deployment* phases.

Describing Service Implementations using IEC 61499

Once `Functions` can be implemented, they must be able to exchange information. This means that `ServiceImplementations` of `Functions` need a representation in IEC 61499. Furthermore, the `Connections` between `ServiceImplementations` also need a representation in IEC 61499. After the `Functions` of an `Application` have been assigned to a `Resource` with a `FunctionMapping`, there is a difference between internal and external `Connections`. Internal `Connections` connect `Functions` in the same `Resource`. External `Connections` connect `Functions` in different `Resources`.

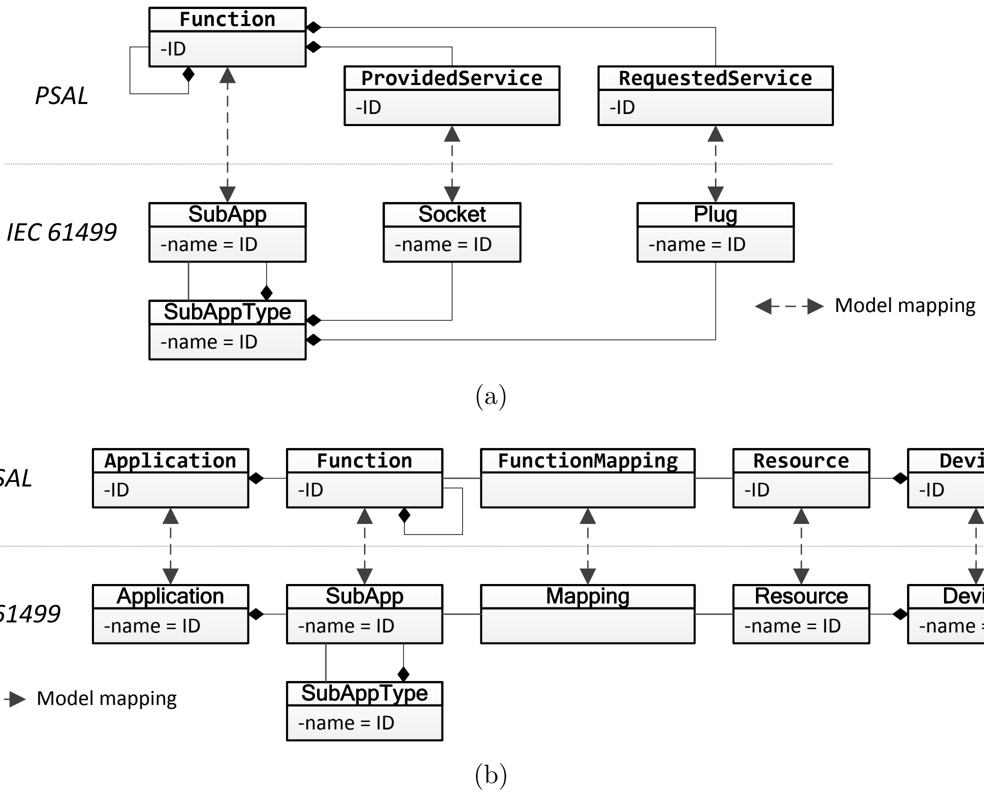
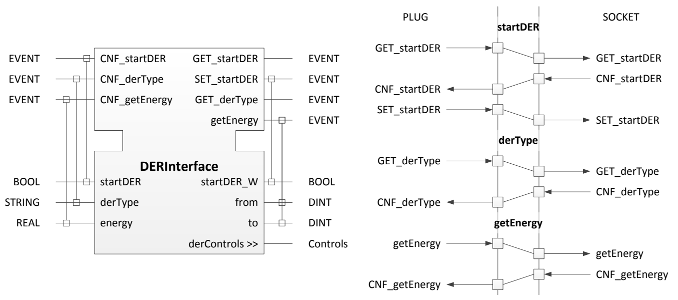


Figure 4.10: Mapping between IEC 61499 and PSAL: (a) Detailed mapping between `Functions` and SubApps[499]; (b) Mapping for application and system models.

As seen in Figure 4.10a, the `ProvidedServices` of a `Function` are implemented as Sockets[499] and the `RequestedServices` are implemented as Plugs[499]. Each Plug[499] and Socket[499] is defined by an AdapterType[499]. Since each `ServiceImplementation` is defined by either an `Interface` or an `Event`, both of these need a mapping to an AdapterType[499] representation in IEC 61499.

Figure 4.11 shows an example, where an `Interface` is mapped to a corresponding AdapterType[499] in IEC 61499. The AdapterType[499] in Figure 4.11b is defined using the Socket[499] representation shown in Figure 2.5. The `Interface` contains `Attributes` and `Operations`. The type of an `Attribute` can either be a simple type or an already defined type (e.g., another `Interface`). All `Attributes` that are of an already defined type are represented as a Plug[499] interface on the AdapterType[499], see the *derControls* Plug[499] in Figure 4.11b. `Attributes` that have a simple type are transformed into a set of events and data ports. All `Attributes` in PSAL are automatically assigned a get-function. Non-`readonly` attributes are also assigned a set-function. To represent this for the AdapterType[499] a GET-Event[499] is created for each `Attribute`, and a SET-Event[499] for non-`readonly` `Attributes`, see Figure 4.11. `Operations` in PSAL, on the other hand, define a callable function, and can have both inputs (notated with `in`), outputs (notated with `out`), and a return value. The `Operation` *getEnergy*, defined in Figure 4.11a, is mapped to an Event[499] with the same name as the `Operation`. This event is associated with two data ports: *from* and *to* in Figure 4.11b. Once the `Operation` has finished a CNF-Event[499] is triggered.

```
interface DERInterface {
  attribute Controls derControls
  attribute boolean startDER
  readonly attribute string derType
  void getEnergy(in int32 from,
    in int32 to,
    out float32 energy)
}
```

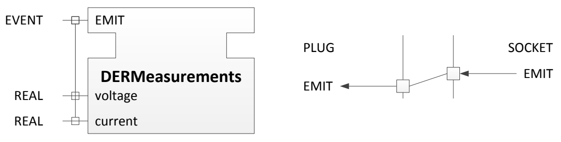


(a) (b)

Figure 4.11: Mapping between an `Interface` and an AdapterType[499]: (a) `Interface` definition in PSAL; (b) AdapterType[499] definition in IEC 61499.

The `Events` of PSAL are also mapped to AdapterTypes[499], see Figure 4.12. Since an `Event` only represents a one-way information exchange its implementation as AdapterType[499] is simplified, with only one Event[499]. The `StateMembers` of the `Event` are mapped to data ports of the AdapterType[499].

```
eventtype DERMeasurements {
  public float32 voltage
  public float32 current
}
```



(a) (b)

Figure 4.12: Mapping between an `Event` and an AdapterType[499]: (a) Definition of an `Event` in PSAL; (b) AdapterType[499] definition in IEC 61499.

In IEC 61499, connections between FBs[499] that are mapped to different Resources[499] may need extra attention during deployment. These connections need to be split up using explicit communication FBs[499]. The IEC 61499 standard provides two generic communication patterns: client/server for bidirectional connections and publish/subscribe for unidirectional connections [64]. The service types in PSAL can also be mapped to the same patterns. The `Interface` models a typical client/server connection, where the client calls the `Operations` of an `Interface` and gets/sets the `Attributes`. In order to model publish/subscribe relationships, the `Event` can be used. An `Event` is emitted by a publisher, and consumed by a subscriber. Figure 4.13 illustrates how two connections are split up into client/server FBs[499].

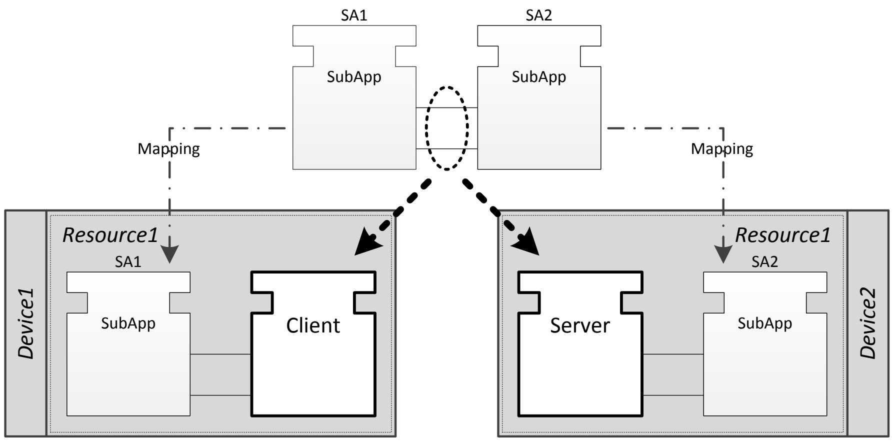


Figure 4.13: Splitting connections into communication FBs[499].

4.4.3 Using Different Communication Protocols

Once the `ServiceImplementations`, the `Interfaces`, and the `Events` can be implemented, for both internal and external connections, different information models can be mapped to the PSAL information model (i.e., to `Interfaces` and `Events`). This allows the user to model communication with different protocols (e.g., IEC 61850, Modbus, OPC UA) and directly include this into the functional model of the application.

Mapping IEC 61850 to Interfaces and Events in PSAL

As an example of how information models can be mapped to the PSAL generic information model, IEC 61850 is used. The IEC 61850 standard defines LN classes for multiple applications. In the standard, these classes are defined as tables. Figure 4.14a shows an excerpt of the *MMXU* class from [66]. Moreover, IEC 61850 also provides the SCL description language. It is based on XML, and is used to write configuration files for IEDs. In order to distinguish IEC 61850 elements defined in SCL they are emphasized with a slanted font (e.g., *LDevice*^{850}).

If an SCL file is configured for an IED, it specifies which LN classes are provided by this IED. For each provided LN the SCL file also includes a type definition—an *LNodeType*^{850}. Since data objects of an LN class can be optional, the *LNodeType*^{850} defines which of these are used in the current configuration. Figure 4.14b shows the SCL definition of *MMXU*. For the *DAType*^{850}, called *AnalogueValue*, only one of the optional attributes is used—the attribute *f*.

With SCL as foundation, a mapping from IEC 61850 to PSAL can be formulated. Apart from the *LNodeType*^{850}, SCL uses *DOTypes*^{850} for CDCs, and *DATypes*^{850} for DA types. For these three types the same mapping approach can be used—each type instance is mapped to an `Interface`. Each type in IEC 61850 contains objects: Data Objects

MMXU class				
Data object name	Comon data class	Explanation	T	M/O/C
Measured and metered values				
TotW	***MV***	Total Active Power		O
PNV	WYE	Phase to neutral voltage		O

MV class					
Data attribute name	Type	FC	TrgOp	Value	M/O/C
Data attribute					
mag	***AnalogueValue***	MX	dchg, dupd		M
q	Quality	MX	qchg		M
t	TimeStamp	MX			M

AnalogueValue type definition			
Attribute name	Type	Value/Value range	M/O/C
i	INT32	integer value	O
f	FLOAT32	floating point value	O

(a)

```
<LNodeType lnClass="MMXU" id="MMXU">
  <DO name="TotW" type="MV"/>
  <DO name="PNV" type="WYE"/>
</LNodeType>
<DOType id="MV" cdc="MV">
  <DA name="mag" type="AnalogueValue"
       bType="Struct" fc="MX"/>
  <DA name="q" bType="Quality"
       fc="MX"/>
  <DA name="t" bType="Timestamp"
       fc="MX"/>
</DOType>
<DAType id="AnalogueValue">
  <BDA name="f" bType="FLOAT32"/>
</DAType>
```

(b)

```
abstract interface MMXU {
  attribute MV TotW
  attribute WYE PNV
}
abstract interface MV {
  readonly attribute ⤶
       ↳ AnalogueValue mag
  readonly attribute uint16 q
  readonly attribute date t
}

abstract interface ⤶
     ↳ AnalogueValue {
  attribute float32 f
}
```

(c)

Figure 4.14: Mapping between an *LN*^{850} and an `Interface`: (a) Excerpt of the *LN*^{850} *MMXU* in IEC 61850; (b) *MMXU* as SCL description; (c) *MMXU* as `Interface` description.

(*DOs*^{850}), *DAs*^{850}, or Basic Data Attribute (*BDAs*^{850}). These are mapped to `Attributes` in PSAL. In Figure 4.14c, the PSAL definition of *MMXU* is shown.

With this mapping, all LN classes that are defined in the IEC 61850 standard can be imported and represented in PSAL. The mapping also makes it possible to import a whole SCL file into a corresponding PSAL model. The other way around, if IEC 61850 `Interfaces` are used in a PSAL model (i.e., as in Figure 4.14c), these can also be exported to an SCL file. Such an SCL file is one possible target configuration, as seen in Figure 4.2. Consequently, with the usage of IEC 61499 as implementation language and protocol mappings like IEC 61850 the *implementation* phase can be started.

4.5 Phase II: Implementation

With the selection of the implementation language and integration with PSAL, the next phase of the rapid engineering methodology is the *implementation* phase. The goal of this phase is to provide the specified functions with logic. Since IEC 61499 is used as implementation language this corresponds to finishing the application by implementing the SubAppTypes^{499}, as presented in Section 4.4.2.

The implementation phase contains three main steps: *(i)* transforming the `Application` and `System` models into an Application^{499} and a System^{499} in IEC 61499, *(ii)* automatic generation of functions (i.e., generation of SubAppTypes^{499}), and *(iii)* definition of functions by the user (i.e., the user implements the SubAppTypes^{499}). An overview of the *implementation* phase is seen in Figure 4.15.

Based on the design done by the user with PSAL in the previous phase, a transformation to IEC 61499 is possible. This transformation uses the rules defined in Section 4.4.2.

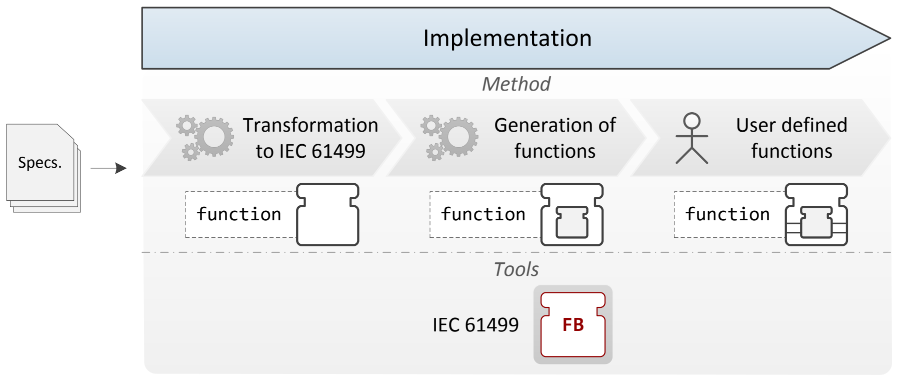


Figure 4.15: Overview of the different steps of the *implementation* phase.

Consequently, the first step is to transform the `Application` and `System` into equivalent IEC 61499 Application[499] and System[499] models. To do this, the mapping from Figure 4.10 is used. The Application[499] is also filled with one SubApp[499] for each Function[499] defined with PSAL. This step also generates empty SubAppTypes[499] for each SubApp[499]. This step contributes to the realization of requirement R5-Seamlessness, and also to requirement R6-Rapidness and Effort. It allows a seamless, and therefore rapid, transition between the *design* phase and the *implementation* phase, which reduces the manual effort required by the user.

The next step is an automated generation of the logic within the SubAppTypes[499]. This generation is based on previous user implementations. Each specified `Function` is implemented as a SubAppType[499]. If all implementations are saved, they can be used as templates for future implementations. By automatically comparing the design of a new `Function` with an already existing `Function` their similarity can be measured. If this similarity is high enough the already existing implementation can be used for the new `Function` as well. After the generation, the copied implementation can be adapted to fit the specific needs of the new `Function`. This step also contributes to requirement R6-Rapidness and Effort.

The third, and last step of the *implementation* phase, is for the user to complete the implementation of the `Functions`. On the one side, many of the generated implementations may need adjustments or final adaptations. On the other side, the SubAppTypes[499] that were not generated need to be implemented. The user implements a SubAppType[499] by adding to it other FBs[499] and SubApps[499]. These are connected with each other as well as with the interfaces of the SubAppType[499]. The last two steps of the *implementation* enable the realization of requirement R2-Implementation.

In many cases, only some `Functions` need to be implemented since the other `Functions` are provided by another actor or they already exist in the system. These other `Functions` do not need to be implemented. Instead, their design will be used to configure communication interfaces and to integrate the new `Functions`. This also allows the integration with legacy systems, see requirement R9-Handling Legacy Systems.

4.5.1 Automatic Implementation of Functions

As seen in Figure 4.15, the second step of the *implementation* phase is an automated generation of `Functions`. The goal with this step is to support the user with the creation of implementations (i.e., SubAppTypes[499]) for the specified `Functions`. There are different approaches for an automated creation of such implementations. One possibility is to create a semantic mapping between `Interfaces`/`Events` and their implementation. For example, the *MMXU* LN presented in Figure 4.14a can be associated with a SubAppType[499] for implementation. Whenever the *MMXU* `Interface` is used by a `ServiceImplementation` in a `Function`, the associated `SubAppType` can be used for implementation of the `Function`. Of course, it must also be possible for the user to define their own semantic mappings.

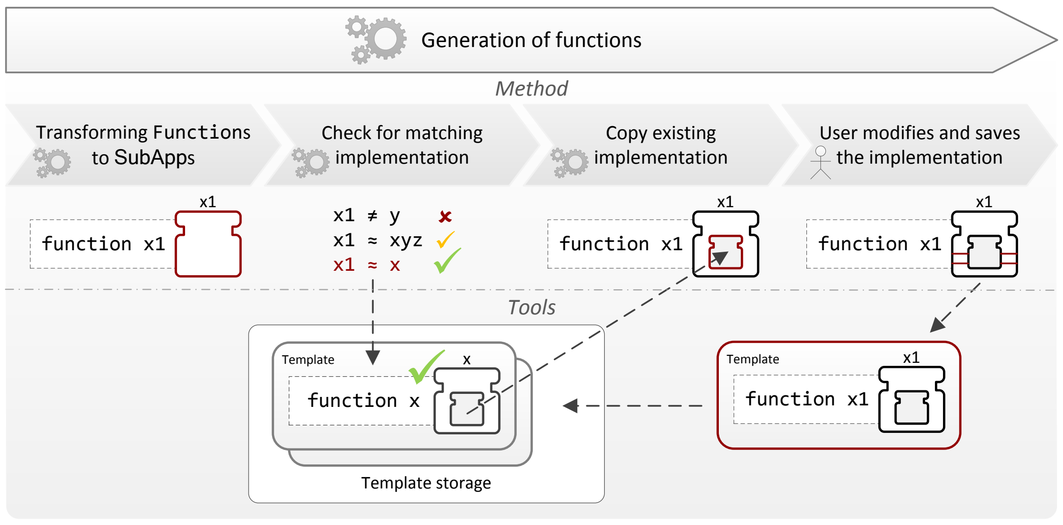
Often it is not possible to define a distinct implementation for an `Interface` or `Event`. Furthermore, in many cases platform-specific parts are also needed to create a fully-functional implementation. For example, the *MMXU* is LN that provides measurements. Consequently, voltage and current sensors need to be interfaced for the implementation. Such cases always require platform-specific knowledge.

In this thesis, an approach is used that combines the possibility to create semantic mappings and also uses a simple machine learning of user implementations. An overview of the approach is shown in Figure 4.16. The main tool for the automated generation is a *template storage*. A *template* represents a previous user implementation and is a combination of a `Function` design and a `SubAppType`. After the initial step of the *implementation* phase, an empty SubAppType[499] is generated for each specified `Function`. In order to determine if a similar implementation already exists, the `Functions` of the new `Application` are compared to the stored *templates*. When a match is found, the user is asked if the previous SubAppType[499] should be used for the new `Function` as well. If the user accepts, the implementation of the old SubAppType[499] is copied to the new `SubAppType`. Finally, the user has the possibility to adapt the new `SubAppType` and make necessary changes, if there are any. Once the user saves the `SubAppType` a new *template* is created and stored.

In order to check if a matching implementation exists, string metrics are used to compare the `Functions`. The following parts are considered for the comparison:

- The `ID` of a `Function` is an indication of its intended functionality.
- If a comment is provided directly before a `Function` declaration this is also used.
- The `ID` of the containing `Application`.
- Each `ServiceImplementation` is included with `ID` and type.

In the literature there are multiple string metrics for measuring the edit distance between two strings [85, 80]. Principally they measure the minimum number of operations needed



Generation of functions
Method
Transforming Functions to SubApps
Check for matching implementation
Copy existing implementation
User modifies and saves the implementation
x1
function x1
x1 ≠ y
x1 ≈ xyz
x1 ≈ x
Tools
Template
function x
x
Template storage
Template
function x1
x1

Figure 4.16: Automated generation of SubAppTypes[499] for Functions.

in order to transform one string into another. This can also be utilized to measure the similarity of two strings. Consequently, if the similarity of two Functions is measured, for the parts listed in the bullet points above, it can be used as an indication for how similar the functionality is as well. The following metric is used to calculate the similarity s between two Functions

$$\begin{aligned} &s = k_A A + k_F F + k_C C + k_P P + k_R R \quad \text{where} \\ &k_A + k_F + k_C + k_P + k_R = 1 \end{aligned} \tag{4.1}$$

k_A, k_F, k_C, k_P, and k_R are weighting factors and should have a sum equal to 1. They are combined with A, which is the ID of the containing Application, F, which is the ID of the Function, and C, any comment provided directly before the Function. P is a concatenation of the IDs of the ProvidedServices and the IDs of the service type (i.e., the ID of the provided Interface or Event). R is the same concatenation, but for the RequestedServices.

If the similarity s between two Functions is above a certain threshold they are considered matching. When multiple *templates* are available, there might be multiple matches as well. In such cases the user decides which implementation should be used. Using this approach each user implementation is also available as a template for future applications. This also allows an easier start for new users, since they can take advantage of designs made by more experienced users. Furthermore, the *template storage* can also be filled with predefined solutions (e.g., for IEC 61850 Interfaces). This simple approach can also be extended to include more advanced metrics of the user design decisions.

4.6 Phase III–IV: Validation and Deployment

The next phase is the *validation*. The main goal with this phase is to validate and test the implemented functions and algorithms that were developed in the previous phase. The next step is the to *deploy* the developed application be to the field, but only after a successful validation. However, with the choice of IEC 61499 as implementation language the validation and the deployment phases melt together. In order to execute an IEC 61499 Application^{499} the deployment guidelines defined in Section 4.4.1 should be followed. Summarized, they specify the following process:

- Mapping of SubApps^{499} and FBs^{499} to their corresponding Resources^{499}.
- Adding platform-specific information to the Resource^{499}.
- Configuration of the actual field instance of the Device^{499} according to the contents of its Resources^{499} (i.e., the field Device^{499} is set up to perform the tasks defined by its Resources^{499}).

This process is followed independently of if the Application^{499} is validated or if it is executed in a field environment. The only difference is that the platform-specific information will differ depending on the execution platform. Consequently, the actual implemented functionality remains the same independently of where it is executed [7, 4, 150]. Therefore, the *validation* and *deployment* phases can be combined. This is however not always the case. For other implementation languages, a strict separation of the validation and the deployment may be desired. For example, if a modeling and simulation language (e.g., Modelica, Matlab/Simulink) is used as implementation language, the validation could be done directly using simulations, and code generation (e.g., to C code) could be one possibility for the deployment.

An overview of the combined *validation and deployment* phase is seen in Figure 4.17. It contains three main steps: *(i)* generation of platform-specific code, *(ii)* deployment of the code to the dedicated platform, and *(iii)* the actual execution and/or validation. Automated support is provided for the first step, whereas the other steps are mainly carried out by the user.

The first step is the generation of a platform-specific implementation based on the design and the platform-independent implementation from the previous phase. Three types of code are automatically generated. First, platform-specific information is added to the Resources^{499}. This includes generation of SIFBs for communication between Devices^{499}. Secondly, configurations are generated to setup the communication SIFBs. Finally, simulation models are generated from the `System` specification for simulative validations.

Once the necessary platform-specific code is generated it is deployed to its execution platform. This can be either field devices, a simulation tool or a combination thereof. In this work, simulation models are generated as CIM models. Thus, they can be simulated on any power system simulation tool with support for the CIM description format.

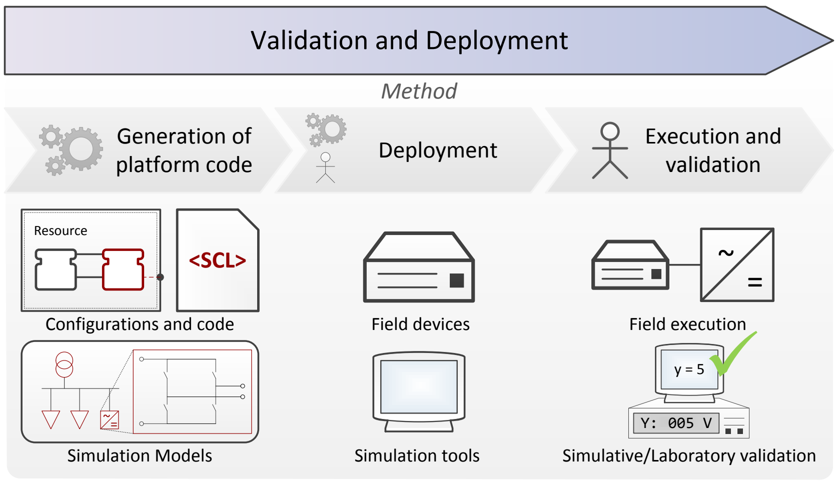


Figure 4.17: Overview of the steps for the combined *validation and deployment* phase.

The final step is the actual execution and/or validation of the developed application. The validation can either be simulative or a laboratory test. How the validation is carried out is decided by the user and is not a focus of this thesis. The main focus is instead to provide an automated support for transferring the implemented application to the desired platform: field devices for automation functions and simulation tool for generated models. With the combined *validation and deployment* phase the requirements R3-Testing and Validation and R4-Release and Deployment are realized.

4.6.1 Generation of Platform Code

The first step of the *validation and deployment* phase concerns the generation of platform-specific code. In this thesis, mainly three types are generated: platform-specific IEC 61499 implementations, communication configurations, and simulation models. The following sections describe this process in detail.

Platform-Specific IEC 61499 Implementation

The Application[499] that is the result of the *implementation* phase is platform-independent. This means that no platform-specific information is provided within the Application[499]. Before it can be deployed, this information must be added in order to be executable on its target platform. Different platforms must be supported. It can be either a simulation platform, a laboratory platform, or a field device. The platform-specific information is added to the Resources[499] where the SubApps[499] are distributed. The information is on the one hand SIFBs for the communication between Devices[499] and on the other hand necessary FBs[499] for the process interfaces (e.g., for field sensors and actuators).

This work is mainly focused on the generation of SIFBs for the communication between SubApps[499]. Each connection that represents a network communication between two

Devices[499] is split up and SIFBs are added to the corresponding Resources[499], as seen in Figure 4.13. Since each connection is associated with an `Interface` or `Event` this information is used for the generation. If the connection is defined by an `Interface` client/server SIFBs are generated. The server FB[499] is created in the Resource[499] of the providing SubApp[499] (i.e., the SubApp[499] with the Socket[499] interface). Similarly the client FB[499] is created in the Resource[499] of the requesting SubApp[499] (i.e., the SubApp[499] with the Plug[499] interface). For `Event` connections publish/subscriber SIFBs are generated. In this case, the publisher FB[499] is created in the Resource[499] of providing SubApp[499] and the subscriber FB[499] is created in the Resource[499] of the requesting SubApp[499]. Depending on the protocol that is used, the SIFBs are properly configured. How this configuration is done depends on the tool and platform that is used. The platform-specific configuration is discussed in more detail in Section 5.3.

The generation of interfaces to sensors and actuators is not directly in the focus of this thesis. In modern power systems, many of the sensors and actuators also provide access through Ethernet interfaces. Therefore, the same approach as for the communication connections between Devices[499] can be used to access the low-level field equipment. Any SubApps[499] belonging to already existing `Functions` do not need to handled during the platform-specific implementation. Since they are already implemented in the field there is also no need to generate any communication SIFBs in their Resources[499]. To prevent this, these SubApps[499] should be unmapped before generation. These opportunities also allow the interaction with existing system components and enables the realization of R9-Handling Legacy Systems.

Communication Configurations

Generally different types of configurations can be generated. On one hand, and as already discussed, it is possible to generate devices configurations in order to configure their communication stacks. Examples are the already discussed SCL files but also configuration of IP-addresses etc. On the other hand, since not only `Devices` are specified in PSAL, but also the communication infrastructure and `Connections`, it is also possible to create configurations for these as well. One possibility could be to take advantage of SDN controllers. By generating configurations for these controllers, they can be exploited to configure the whole communication network [118]. In this thesis, the first possibility is used to show the feasibility of automatic creation of communication configurations.

IEC 61850 is using SCL files in order to configure the information flow between IEDs as well as network properties (e.g., VLAN identifier or priority). Based on the mapping between PSAL and IEC 61850 (see Figure 4.14 and Section 4.4.3) such SCL files can be created for each `Device`. In order to support this, the mapping in Figure 4.14 is extended with the mappings in Figure 4.18.

As seen in Figure 4.18a, the `Device` is mapped to the *IED*[850] and from each `Resource` a *LDevice*[850] is created. In an SCL file, *LDevices*[850] contain LN instances (i.e., *LN*s[850]). In PSAL, these LN instances are represented by the `ServiceImplementations`, which

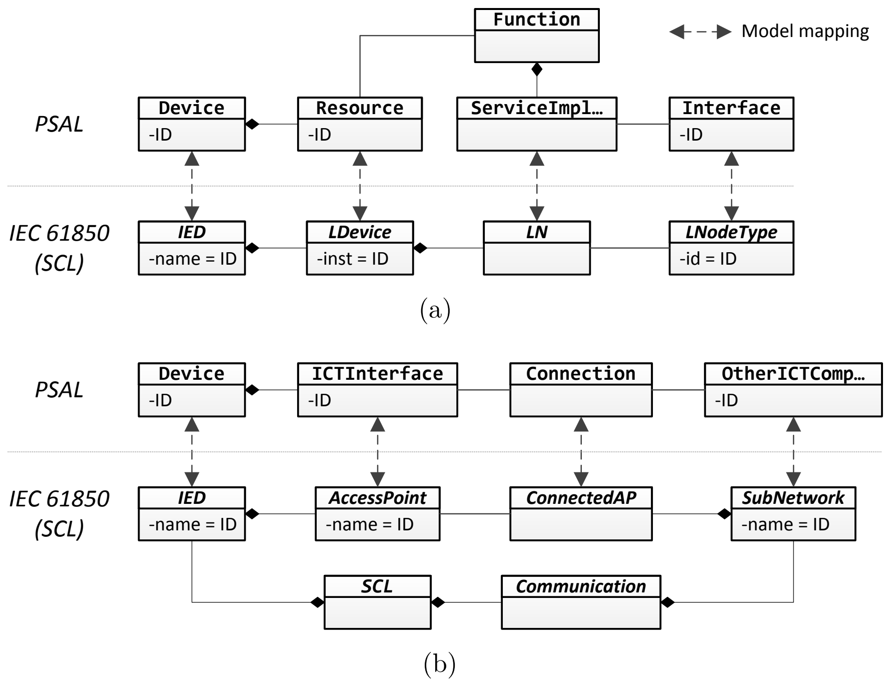


Figure 4.18: Mapping between PSAL and IEC 61850: (a) Mapping between `ServiceImplementations` and *LN*s^{850}; (b) Mapping between the communication description in PSAL and in SCL.

in turn are related to a `Resource` if their implementing `Function` is associated with the `Resource` through a `FunctionMapping`. Finally, each `ServiceImplementation` is defined by an `Interface`, which is mapped to the *LNodeType*^{850}. This is also the mapping that was presented in Section 4.4.3.

In Figure 4.18b it is shown how the communication specification parts of PSAL are mapped to the corresponding parts in SCL. Each `ICTInterface` of a `Devie` is mapped to an *AccessPont*^{850}. Moreover if `OtherICTComponents` like `gateway`, `switch`, or `router` are used these are mapped to a *SubNetwork*^{850} in SCL. The `Connections` between `ICTInterfaces` and `OtherICTComponents` are represented by a *ConnectedAP*^{850} in IEC 61850.

There are some cases where the IEC 61850 model is ambiguous. For example, the *AnalogueValue* `DAType` shown in Figure 4.14a has two optional attributes. It is only allowed to use one of these attributes and it is up to the engineer to decide which one. This means that it is not possible to create a final SCL file only from the specification made in PSAL. Also the implementation is needed, since it is here that the engineer decides which one of the attributes should be used.

Simulation Code

An important tool for validation of smart grid applications is simulation. This is especially the case when it is not possible to use the real system for tests and validation. Either because tests with the real system can cause damage or because it is not available. This is also the case with smart grid systems, see requirement R3-Testing and Validation. To provide support for simulative validations, the rapid engineering approach allows the user to automatically generate simulation models, based on the design and implementation.

Using the mapping described in Section 4.3.2, CIM models can be generated from the specification and design in PSAL. Apart from the `ICTComponents` the specification also contains `ElectricalComponents`. As already discussed in Section 4.2.2 and in Section 4.3.2, the electrical component model is based on CIM. Thus it is also possible to generate a CIM model directly from the PSAL specification using the mapping in Figure 4.7. Since the CIM format is supported by a number of power system simulation tools any of these can be used to execute the simulations.

Only the `ElectricalComponents` are transformed into CIM. The `ICTComponents` are intended to be implemented using IEC 61499 also for the simulative validation. Instead of generating simulation models of the `ICTComponents` the IEC 61499 implementations are integrated with the power system simulation using a co-simulation approach [150, 143, 140]. This has the advantage that the implemented code does not change between validation and deployment. Any `ICTComponents` that are not implemented must be manually integrated into the simulation.

4.6.2 Deployment and Execution

The final step of the *validation and deployment* is to validate and execute the developed application. Depending on the validation method, or execution approach, different steps are needed in order to deploy and execute the generated code. Of course these steps are depending on the use case, as well as the different tools used for the development. Nonetheless, some general steps are always followed. Especially the deployment of the generated code is one step that is important for both the validation process as well as the execution phase. As presented above, three types of code are generated: IEC 61499 code, communication configurations, and simulation code. Consequently, all three types must also be deployed.

The process to deploy IEC 61499 code is defined by compliance profiles, as specified in the IEC 61499 standard [64]. One of the parts that such a compliance profile must define is configurability of Devices[499] by the means of software tools. In this work, the "IEC 61499 Compliance Profile for Feasibility Demonstrations" is used [61]. It defines that software tools shall use the management capabilities of Devices[499], and that these capabilities should be available remotely (e.g., via an IP interface). It also defines an XML format for the configuration of the Device[499] from the software tool. The management interface and the XML format are also used for the deployment of the Application[499]. Simply said, the Application[499] is deployed by downloading it to the Device[499]. This deployment process is always the same, independent if it is a validation case or for field deployment.

In the case of communication configurations, a similar approach, as for IEC 61499, is needed for the deployment. Unfortunately, there is no common approach to do this. Instead different communication standards may have their own approaches. In IEC 61850-6, a remote configuration through file transfers is suggested (see Figure 3.1). Once an SCL file has been engineered it is transfered to the IED. This is complemented with a file transfer service defined in IEC 61850-7-2. However, the standard does not define how—and if—an actual reconfiguration of the IED is triggered by such a file transfer. This is specific to the actual implementation [66].

The deployment of simulation code is usually not as problematic as for field installations. A usual scenario is to first import or load the simulation model into a power system simulator. Once the model is loaded the simulation is started within the simulator. Some tools offer support with an API or through other interfaces. For the rapid engineering method in this thesis, no further automatic support is provided beyond the creation of the simulation models, as described in Section 4.6.1. It is the user's responsibility to load the model and start the simulation.

After the deployment, the execution of the application starts. This can be either a validation or a field execution. How the validation is done is decided by the user. The rapid engineering approach in this thesis can support the user with both a simulative as well as an experimental validation. However, it does not cover what is actually validated and what the criteria for a successful validation are. This is dependent on the use case and must therefore be decided by the use case engineer. A field execution is also very much depending on the use case and the current setup. Therefore, the actual process of starting the application must be handled by the user. The rapid engineering process does not provide any automatic support for this part. It is not possible, due to the numerous different setups that would need to be covered.

4.7 Summary and Reflection

In this chapter, a rapid engineering methodology was described. The main goal with the methodology, as stated by the Research Hypothesis, was to reduce the manual work needed by the smart grid engineer. For this purpose, MDE technologies have been applied and although the rapid engineering methodology is focused on applications for power utility automation, it has many similarities with MDE methods for software development.

For example, the MDA approach, already described in Section 2.5.1, can be directly compared to the rapid engineering methodology. The PSAL model developed during the *design* phase can be considered a PIM. It is not related to any specific execution platform. But, it is even more than that. It is also independent of an implementation language. Only after an implementation language was integrated in Section 4.4 could PSAL be mapped to artifacts of this language. Here, the *design* model is called a Implementation Language Independent Model (ILIM). This is seen as the first part in Figure 4.19.

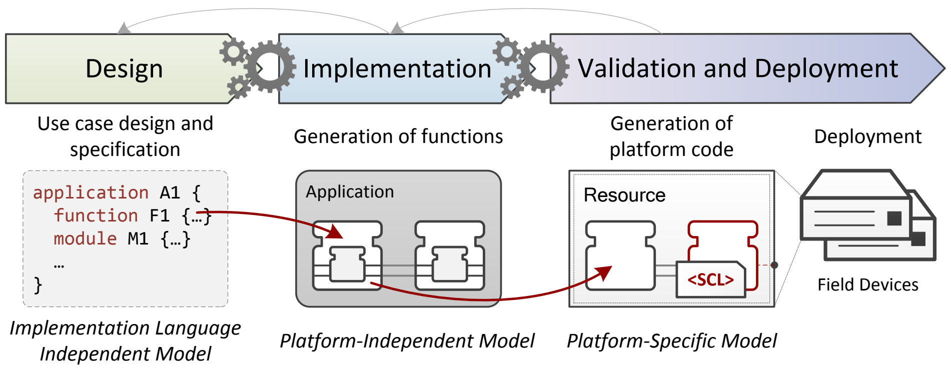


Figure 4.19: Rapid engineering concept with related MDA aspects.

Using the same argumentation as above, the PIM must come after the ILIM. In Figure 4.19, the PIM is seen below the *implementation* phase. The relationship between MDA and IEC 61499 has already been studied by Zoitl et al. [177]. In their opinion, the Application^{499} model of IEC 61499 is comparable to the PIM of MDA. Here, this would mean that the generated Application^{499} and SubApps^{499} (see Section 4.5) are the PIM.

The next step would be to create the PSM. According to Zoitl et al. this model is created by mapping the FBs of the Application^{499} to hardware (i.e., to Resources^{499} of Devices^{499}). Furthermore, services indicated by SIFBs should be inserted into the Resource^{499}, either manually or by the tool [177]. In the case of this work, communication SIFBs are automatically generated and configured for the Resources^{499}, as described in Section 4.6.1. Consequently, these represent the PSM, as seen under the *validation and deployment* phase in Figure 4.19. A PSM for communication configurations is represented by the final configuration file (e.g., an SCL file as described in Section 4.6.1).

The final step that is mentioned in the MDA approach is to create "executable information systems" from the PSM [100]. Either the PSM is transformed into code (e.g., Java, C++) or a model execution engine exists that can directly execute the PSM. The approach in this work is more related to the latter alternative since the IEC 61499 code is deployed through a download using an XML format, as described in Section 4.6.2.

CHAPTER 5

Prototypical Implementation

In order to show the feasibility of the rapid engineering methodology presented in Chapter 4, a prototypical framework was implemented. A number of goals were defined for the framework. First of all, the main goal of the framework is to use it to answer the main research question of this thesis, and to prove the research hypothesis (see Section 1.2). By using the framework it can be shown that the rapid engineering methodology not only works in theory, but also in practice. Another goal was to create a usable framework existing within one tool. This means that the user should not need to shift between different tools in order to complete the engineering process.

The following sections present the prototypical implementation made for the different phases. Finally, an overview of the complete framework is presented at the end of the chapter. The following work is mainly based on [117], although this thesis provides much more detailed description.

5.1 Implementing the Specification Phase

First of all, PSAL must be implemented such that power utility automation use cases can be modeled. Furthermore, an editor is needed accompanied by a compiler that can interpret the PSAL code. The choice fell on the xText [36] environment for the Eclipse Integrated Development Environment (IDE) [35]. On one side, it allows the definition of DSLs and on the other side it also provides resources to implement an editor and compiler. Figure 5.1 shows a conceptual visualization of the resulting PSAL tool, and the different design steps that it includes.

The figure shows the three main steps of the design phase: design of the `Application`, the `System`, and the `Interfaces/Events`. These are all carried out by the engineer using the PSAL editor. The editor automatically uses syntax highlighting based on the defined grammar. Figure 5.2 shows an IEC 61850 `Interface`, defined in the PSAL editor.

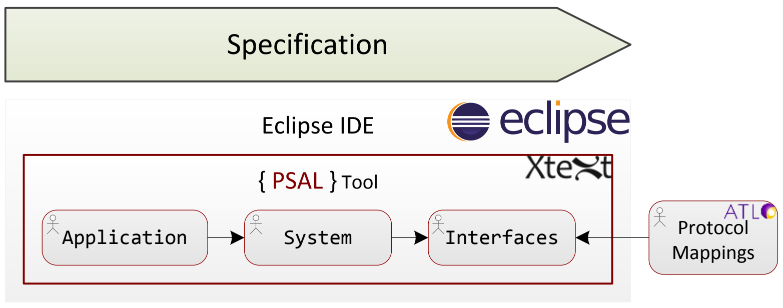
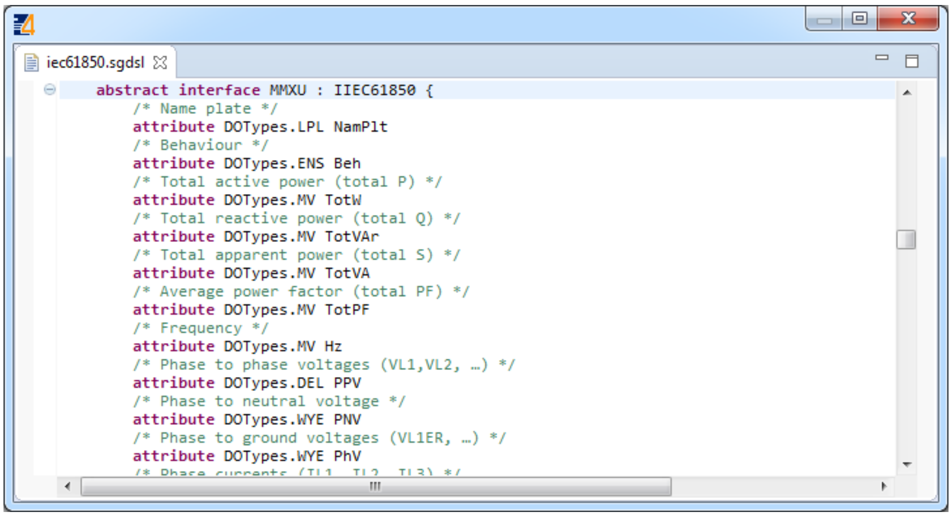


Figure 5.1: Illustration of the PSAL tool implementation.

Additionally, as seen in Figure 5.1, the user can also define protocol mappings, described in Section 4.4.3. For this purpose ATL [78] for the Eclipse IDE was used. ATL is used to describe model-to-model transformations. It allows the user to describe how an existing information model (e.g., IEC 61850) can be mapped to the information model in PSAL. In the next step, this description can be used to import and transform the information model into a resulting PSAL model. Finally, this model can be opened with the PSAL editor. With the IEC 61850 information model as an example, the *MMXU* `Interface` shown in Figure 5.2 is the result of such a transformation. As input, the full SCL description of the *MMXU* from Figure 4.14b is used.



```
abstract interface MMXU : IIEC61850 {
    /* Name plate */
    attribute DOTypes.LPL NamPlt
    /* Behaviour */
    attribute DOTypes.ENS Beh
    /* Total active power (total P) */
    attribute DOTypes.MV TotW
    /* Total reactive power (total Q) */
    attribute DOTypes.MV TotVAr
    /* Total apparent power (total S) */
    attribute DOTypes.MV TotVA
    /* Average power factor (total PF) */
    attribute DOTypes.MV TotPF
    /* Frequency */
    attribute DOTypes.MV Hz
    /* Phase to phase voltages (VL1,VL2, …) */
    attribute DOTypes.DEL PPV
    /* Phase to neutral voltage */
    attribute DOTypes.WYE PNV
    /* Phase to ground voltages (VL1ER, …) */
    attribute DOTypes.WYE PhV
    /* Phase currents (IL1, IL2, IL3) */
```

Figure 5.2: Definition of the *MMXU LNodeType*^{850} in the PSAL editor.

Figure 5.2 also shows how comments in PSAL can be used for documentation purposes. Each comment describes the next `Attribute` and is a direct copy of the description of each *DO*^{850} in the SCL file. Thus, no information is lost during the transformation.

5.2 Prototype for the Implementation Phase

In order to facilitate the implementation phase, PSAL should be extended with IEC 61499 capabilities. This will allow the implementation of PSAL `Functions`. For this purpose, the 4diac-ide [176, 35] was chosen, an open-source IEC 61499 compatible development environment. It is also based on the Eclipse IDE and thus it was possible to create one environment capable of both design and implementations. 4diac also provides a runtime environment for the execution of IEC 61499 control applications. However, in order to support a smooth transition between the *design* phase and the *implementation* phase some additions were needed for the 4diac-ide.

In Figure 5.3, a conceptual representation of the 4diac-ide together with the extensions is shown. First of all, an addition was made to handle the generation of the Application[499] and the System[499] models based on the PSAL description, see Section 4.4.2 and Section 4.5. When the user saves any changes in the PSAL editor, all `Applications` and `Systems` are parsed and transformed into their correspondence in IEC 61499. For this transformation, ATL rules were defined that take the PSAL model as input and create IEC 61499 models as output. These models are then loaded by the 4diac-ide environment.

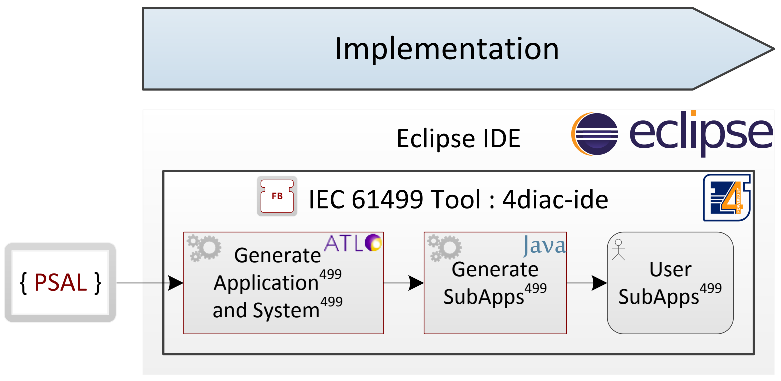


Figure 5.3: Illustration of the IEC 61499 based on the 4diac-ide.

The next step is to generate SubAppTypes[499] and SubApp[499] instances, as described in Section 4.5.1. This addition was created using plain Java and is automatically executed after the generation of the Application[499] and the System[499]. Previous `Function` implementations are saved in a database. Newly specified `Functions` are compared to the ones in the database. If a match is found the user is asked if it should be used for the new implementation as well. If that is the case the old SubAppType[499] is copied and used for the new SubApp[499] instance as well. For `Functions` that do not have a match empty SubAppTypes[499] are generated.

Following the automatic generation of the Application[499], the System[499], and the SubApps[499], manual interaction by the user is needed. On one hand, the empty SubAppTypes[499] must be implemented. On the other hand, the user also has the possibility to modify the generated SubAppTypes[499]. For this step, all the standard functionality of the 4diac-ide is available to the user [35].

5.3 Tools for the Validation and Deployment

As already described in Section 4.6, the *validation and deployment* phase has three main parts: code generation, deployment, and execution/validation. The rapid engineering method provides automation support for the first two parts. In order to accommodate this support, another extension was added to the 4diac-ide to handle the generation of Resource[499] specific FBs[499]. Also, two new features were added to support the generation of communication configurations and simulation models. These additions together with the conceptual realization for the *validation and deployment* phase are shown in Figure 5.4.

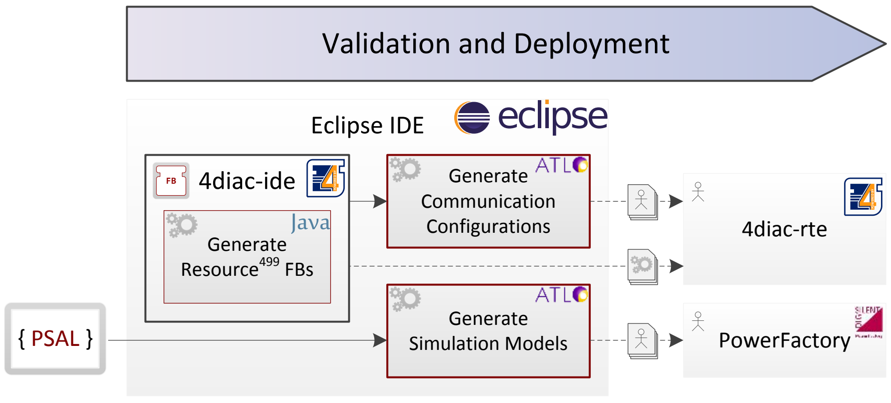


Figure 5.4: Illustration of the tools used for the *validation and deployment* phase.

The first step of the *validation and deployment* is to generate platform specific FBs[499] for the Resource[499]. As already described in Section 4.4.2, connections between FBs[499] that are mapped to different Resources[499] may need extra attention before deployment. In 4diac, this is handled with SIFBs, as seen in Figure 4.13.

For communication, the 4diac-ide provides client/server and publish/subscriber SIFBs. These are also proposed in the IEC 61499 standard [64]. With 4diac, these SIFBs can also be parameterized to use different communication protocols. What protocols can be used are defined by the 4diac-rte (short for 4diac Runtime Environment (RTE) and also known as FORTE), which accompanies the 4diac-ide. The standard protocols used by 4diac are ASN.1 over TCP for client/server connections and ASN.1 over UDP for publish/subscriber connections. However other protocols like Modbus, OPC UA, Message Queue Telemetry Transport (MQTT), and Ethernet PowerLink are also natively supported. For this work, support for IEC 61850 was also implemented using the libiec61850 library [152, 87].

The 4diac-ide also has a primitive support for the automatic generation of communication SIFBs. For connections between FBs[499] in different Resources[499] publish/subscriber SIFBs can be automatically generated [83]. This feature of 4diac was extended during this thesis. Support for automatic generation of client/server based communication, as well as to handle Adapter connections was added to the 4diac-ide. Consequently, for connections based on `Interfaces` client/server FBs[499] are generated, and for connections based on `Events` publish/subscribe FBs[499] are generated.

Furthermore since PSAL offers a better possibility to describe communication interfaces, this information can also be used to automatically configure the SIFBs. Figure 5.5 extends Figure 4.13 by also showing how the communication SIFBs are parameterized based on the defining `Interface`. The `Interface` *i1* inherits from an IEC 61850 LN called *MMXU* (see Figure 4.14 for more information on the definition of *MMXU*). As a result, both the client FB[499] and the server FB[499] are configured to use IEC 61850 as communication protocol. `Interfaces` and `Events` without inheritance are automatically configured to use the standard protocol ASN.1. Sometimes it is also possible to override the choice of protocol in the `ServiceImplementation` definition. With help of the `ProtocolUsage` part it may be possible to define another communication protocol than the one defined by the `Interface` or `Event`. For example, it is always possible to use ASN.1, since it does not provide any more semantic than the data type, but IEC 61850 can only be used if the `Interface` also inherits from an IEC 61850 `Interface`.

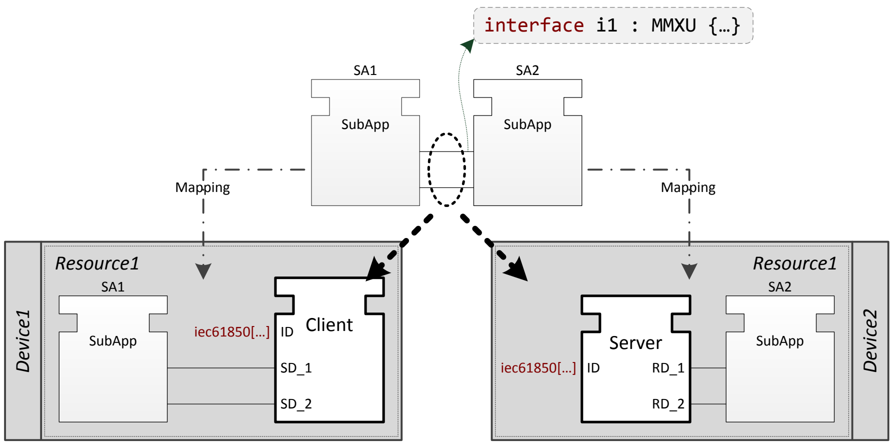


Figure 5.5: Automatic generation of communication FBs[499] based on `Interfaces`/`Events`.

Accompanying the 4diac-ide is the 4diac-rte. It allows the execution of IEC 61499 applications. Both 4diac tools support the "IEC 61499 Compliance Profile for Feasibility Demonstrations". Consequently, the deployment of the Resources[499] to their Devices[499] is automated, and results to a simple download from the 4diac-ide to 4diac-rte.

In order to validate the generation of communication configurations, the prototypical implementation supports the automatic generation of SCL files based on used IEC 61850 `Interfaces`. To achieve this a transformation between PSAL and SCL was implemented using ATL. The transformation uses the mapping described in Section 4.6.1. As seen in Figure 5.2, imported IEC 61850 `Interfaces` include all possible *DOs*[850], mandatory and optional. In order to decide which *DOs*[850] should be used in the final SCL configuration the implemented SubAppTypes[499] for the application are examined. Only the *DOs*[850] that are actually used in the SubAppTypes[499] will also be used in the final SCL configuration. The resulting SCL file must be manually transfered to the Devices[499] where the IEC 61850 client and server FBs[499] are executed.

In Section 4.6.1, it is described how CIM models can be generated based on the `ElectricalComponents` in the PSAL description. For the validation in this work, PowerFactory is used as power system simulation tool [112]. It also supports CIM, and it is possible to import CIM models into PowerFactory. Thus, a transformation was added to the prototypical implementation that creates CIM models compatible with PowerFactory. As with the other transformations ATL was used for the realization.

5.4 Resulting Framework

By connecting the parts above together a resulting framework is created, which is the prototypical implementation for the rapid engineering methodology developed in this thesis. The framework is running within one Eclipse instance. This means that the user does not have to switch between different tools during the development. Other tools are only needed for the validation and the execution. In Figure 5.6, the complete framework is shown. The figure also shows the intended workflow, and if user interaction is needed or if automation is provided. A red border indicates that this is a new part, created for this thesis, and a black border marks already existing parts.

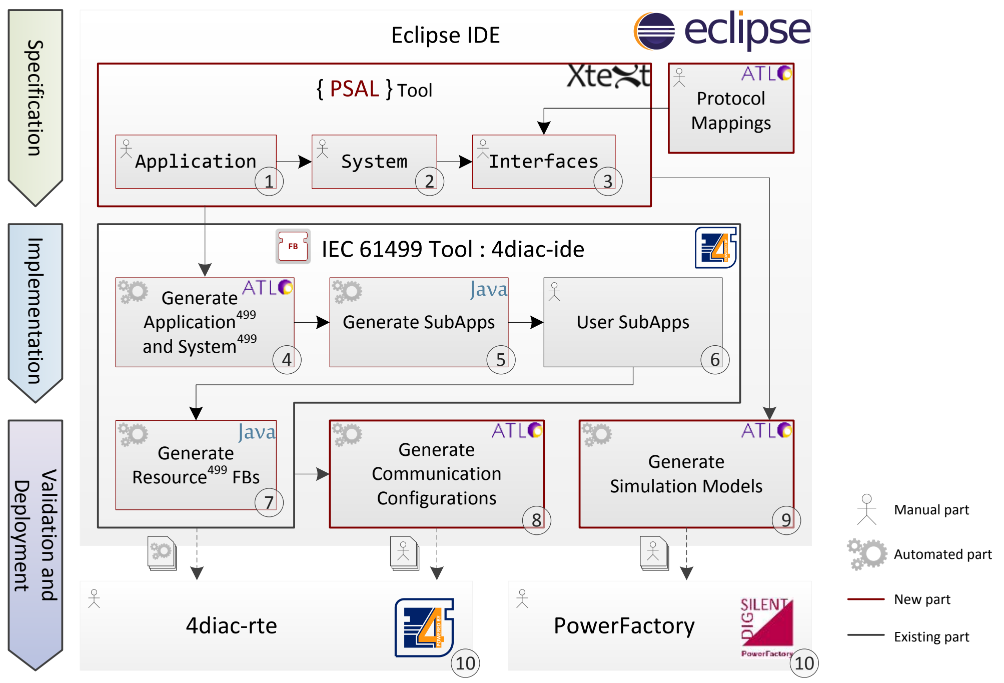


Figure 5.6: Prototypical implementation of the rapid engineering methodology.

CHAPTER 6

Applying the Rapid Engineering Methodology

As introduced in Section 1.3, the scientific research method used in this work contains five steps, where the fourth step regards the implementation of the formulated theoretical solution. The first part of that step is the actual prototypical implementation of the rapid engineering methodology presented in the previous Chapter 5. However, in this research case, the fourth step also includes the application of the developed prototype. In other words, the rapid engineering method is applied to a selected test case to show how it can be used to develop smart grid applications. For each phase of the engineering process, this chapter also presents how the prototypical implementation (presented in Chapter 5) is used. The main parts of this chapter were already presented in [117].

6.1 Introduction and Use Case Analysis

In order to show how the rapid engineering methodology is used, a use case example is modeled and implemented. The chosen use case considers a coordinated volt-VAr control that is commissioned by the DSO. This includes the implementation of a control device owned by the DSO as well as the implementation of new functions in a DER component. The new DER functionality is commissioned from the DER manufacturer. The use case was selected to cover at least one part of all three business cases presented in Section 3.1.

The component layer of the use case is seen in Figure 6.1. Its structure is similar to the “Integrated Volt VAr Control Centralized” [52] use case presented in BC1. A typical scenario for distribution network operation is used, where the primary goal is to keep network voltage within the allowed limits. To achieve this, local voltage control is provided by selected DERs throughout the network. These DERs affect the voltage at their point of coupling through the use of volt-VAr control [71]. Additionally, at the operation level, a central volt-VAr optimizer supervises and manages the local DER controllers in order to achieve an optimal power level within the whole distribution network.

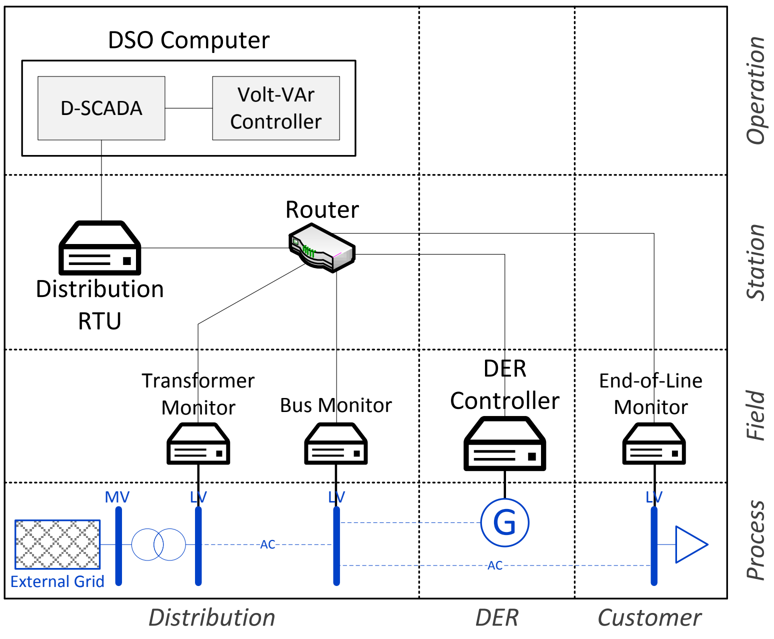


Figure 6.1: Component layer for the test case.

The devices in the field (i.e., the *Transformer Monitor*, the *Bus Monitor*, the *DER Controller*, and the *End-of-Line Monitor*) send measurements to the *Distribution Remote Terminal Unit (RTU)*. The *Distribution RTU* aggregates this information and forwards it to the *DSO SCADA (D-SCADA)* system. Thereupon, the *D-SCADA* forwards all vital information to the *Volt-VAr Controller*. It computes new volt-VAr set points for the *DER Controllers*. The new set points are forwarded by the *D-SCADA* and the *Distribution RTU* to the field controllers. With the volt-VAr set points, the *DER Controllers* can calculate a reactive power set point for the *DER Generator*, illustrated as the circle with the *G* in Figure 6.1. A complete use case description, according to IEC 62559 [69], is provided in Appendix B.4.

For the implementation of this use case, it is assumed that the *D-SCADA* and the *Distribution RTU* are already implemented and installed in the system. From an engineering point of view, both are realized in the same manner as the *Volt-VAr Controller*. For the field components, only a local volt-VAr control functionality for the *DER Controller* needs to be implemented. It is assumed that the monitors are already installed in the system.

In order to properly validate the prototypical implementation of the rapid engineering methodology, this use case covers parts of all three business cases in Section 3.1. The design and specification of the use case cover BC1. The implementation of the *Volt-VAr*

Controller and communication setup of the application is associated with BC2. Finally, the implementation of the *DER Controller* is associated with BC3.

In the following sections, the engineering of the use case, according to the rapid engineering method from Chapter 4, will be presented. First, the whole use case will be designed and specified using PSAL. Secondly, two implementations will be shown in detail: the *Volt-VAr Controller* and the *DER Controller*. Finally, the implementation of the use case will be validated. A simulative validation is used to validate the *DER Controller* and a laboratory validation will be used to validate the complete setup.

6.2 Applying Phase I: Design

In this section, the application of the *specification* phase for the use case is presented. Mainly this means that the use case is designed and modeled using PSAL. As shown in Section 4.3, this phase is based on the SGAM method. Each of the SGAM layers are modeled in different steps using PSAL, as seen in Figure 4.5. This is also shown as the first three steps in Figure 5.6. The following sections show the modeling of these steps for the use case, using the prototypical framework.

The design of the use case using PSAL is based on the use case description in [52], which is also partly described in BC1. This shows how high level use case descriptions can be further analyzed and how functionality and information exchange can be extracted. The following sections were already presented in [117].

6.2.1 Business Case and Functional Design

As the first step of the *specification* phase, a business case should be designed and specified. However, as the use case description in Section 6.1 does not provide any direct information about business cases or goals, these will not be covered in the modeling. Instead, the first step is the functional specification.

According to the definition of PSAL, both the business and the function layers are modeled within the `Application`. For this use case, the `Application` is named according to the use case identification in Table B.37. The `Functions` of the use case are directly derived from the use case actors in Table B.46. These actors are all logical actors and thus each actor represents a certain function in the system. Hence, for each actor a `Function` is declared. In Listing 6.1, a *VoltVArControlCoordinated* `Application` is defined containing a `Function` for each use case actor.

```
application VoltVArControlCoordinated {
  /* Volt-VAr Controller: Controller located in the substation monitoring and
   * controlling the devices in the network. It optimizes the voltage level in
   * the grid by issuing new reactive power set points to the field components.
   */
  @Distribution @Operation
  function VoltVArController {}
  @Distribution @Operation
  function DSCADA {}
  @Distribution @Station
  function DistributionRTU {}
  @DER @Field
  function DERController {}
  @DER @Process
  function DERGenerator {}
  @Distribution @Field
  function TransformerMonitor {}
  @Distribution @Field
  function BusMonitor {}
  @Customer @Field
  function EndOfLineMonitor {}
}
```

Listing 6.1: Initial `Application` definition and declaration of `Functions` for the example use case.

The listing also shows how annotations are used to place each `Function` in a domain and zone. Furthermore the description of the *Volt-VAr Controller* actor from Table B.46 is added as a comment to describe the *VoltVArController* `Function`. In this way, all information from the use case analysis can be included in the PSAL model.

The functional specification also includes a first draft of the information exchange between the `Functions`. Therefore, `Interfaces` and `Events` are defined. At this point of the design their goal is to provide a generic representation of the information that is exchanged in the use case. The formal data model for the information is defined in the design of the information layer. In this case, a first analysis of the exchanged information has already been made based on the use case scenarios. The information exchanged is available in Table B.54. By analyzing how the information is exchanged, `Interfaces` and `Events` can be identified. Any exchange that matches a client/server pattern should be modeled with an `Interface`. Naturally, event based exchanges are modeled with `Events`. Once the `Interfaces` and `Events` have been identified, they are grouped by the engineer into `Modules`. Listing 6.2 shows the identified `Interfaces` and `Event` for the use case.

```
@@ -17,2 +17,27 @@
   function EndOfLineMonitor {}
+  module Measurements {
+    interface GridMeasurement {
+      /* Arithmetic average of the phase to phase voltage for 3 phases */
+      readonly attribute float32 voltage
+    }
+    eventtype AggregatedMeasurement {
+      /* Aggregated measurements from field devices (p.u.) */
+      public float32 vTM, vBM, vEOLM, vDER
+  }}
+  module DERCtrlInterfaces {
+    interface DERVoltVArCurve {
+      /* Voltage set points for droop curve */
+      attribute float32 v1, v2, v3, v4
+      /* Reactive power set points for droop curve */
+     attribute float32 q1, q2, q3, q4
+   }
+   interface DERDirectControls {
+     /* Reactive power set point in percent */
+     attribute float32 qSetPoint
+  }}
+  module FieldControls {
+   interface DERControls {
+     attribute DERCtrlInterfaces.DERVoltVArCurve derVoltVar
+     attribute DERCtrlInterfaces.DERDirectControls derDirectControl
+  }}
 }
```

Listing 6.2: Identified `Interfaces` and `Events` added to Listing 6.1.

The request and the response of the voltage measurement in Table B.54 is represented by the *voltage* `Attribute`. The exchange of the volt-VAr curve is modeled in a similar way by the *DERVoltVArCurve* `Interface`. Likewise, the *DERDirectControls* `Interface` is defined to represent the reactive power set point. Only one `Event` is used in the use case. According to Step 9 of the step-by-step analysis of the use case in Table B.50, the aggregated voltage measurements are published from the *DistributionRTU* to the *DSCADA*. To represent this exchange, the *AggregatedMeasurement* `Event` is defined. The `Interfaces` and the `Event` are grouped manually in two `Modules`: in *Measurements* all measurements are contained and in the *DERCtrlInterfaces* all `Interfaces` related to the control of the DER are grouped. Finally, the defined information is added to the *VoltVArControlCoordinated* `Application`.

In order for the `Functions` to use the information, they need `ServiceImplementations` (i.e., `ProvidedServices` and `RequestedServices`). Finally, the information exchange between the `Functions` is specified by adding `Connections`. They connect requested with provided `ServiceImplementations`. Listing 6.3 shows how `ServiceImplementations` are added to the *VoltVArController* and the *DSCADA* `Functions`, and how they are connected with each other.

```
@@ -1,7 +1,13 @@
 application VoltVArControlCoordinated {
   @Distribution @Operation
-  function VoltVArController {}
+  function VoltVArController {
+    consumes Measurements.AggregatedMeasurement gridStatus
+    requests DERCtrlInterfaces.DERVoltVArCurve dscadaVoltVArCurve
+  }
   @Distribution @Operation
-  function DSCADA {}
+  function DSCADA {
+    emits Measurements.AggregatedMeasurement gridStatus
+    provides DERCtrlInterfaces.DERVoltVArCurve voltVArCurve
+  }
   @Distribution @Station
   function DistributionRTU {}
@@ -38,6 +44,8 @@
```

```
    module FieldControls {
      interface DERControls {
        attribute DERCtrlInterfaces.DERVoltVArCurve derVoltVar
        attribute DERCtrlInterfaces.DERDirectControls derDirectControl
    }}
+   connect VoltVArController.gridStatus with DSCADA.gridStatus
+   connect VoltVArController.dscadaVoltVArCurve with DSCADA.voltVArCurve
  }
```

Listing 6.3: Adding `ServiceImplementations` and `Connections`.

By combining the initial `Application` definition in Listing 6.1 with the new additions in Listing 6.2 and in Listing 6.3, a complete definition is created. This is presented in Listing 6.9, although without any comments. In Appendix C, a full listing of the `Application` is found (Listing C.1). Apart from the comments, it also includes the `FunctionMappings` between the `Functions` and the `Resources` in the `System` model.

6.2.2 System Design

After the functional specification, the next step of the *specification* phase is the system specification. With PSAL, this step includes the definition of the `System` model, where each component participating in the use case is defined.

The diagram in Table B.45 gives an overview of the different components involved in the use case. In general, two types of components are used: `ICTComponents` and `ElectricalComponents`. The `ICTComponents` almost completely coincide with the `Functions` in Listing 6.9. The only differences are that one `Device` is used to host both the *DSCADA* and the *VoltVArControl* `Functions`, and that the `router` component does not have a related `Function`. Listing 6.4 shows an initial definition of the `System` model and the `ICTComponents`. As with the `Functions` in Listing 6.9, each component is placed in its appropriate domain and zone.

```
system DistributionSystemVV {
  @Distribution @Operation
  device DSOComputer {
    ethernet eth0 {ip = "10.0.0.1"}
    resource SCADA
    resource VoltVAr
  }
  @Distribution @Station
  device DistributionRTU {
    ethernet eth0 {ip = "10.0.0.2"}
    ethernet eth0 {ip = "101.0.0.1"}
    resource RTUResource
  }
  @Distribution @Station
  router StationRouter
  @Distribution @Field
  device TransformerMonitor {
    ethernet eth0 {ip = "101.0.0.2"}
    resource MonitorResource
  }
  @Distribution @Field
  device BusMonitor {
    ethernet eth0 {ip = "101.0.0.3"}
    resource MonitorResource
  }
  @DER @Field
  device DERController {
    ethernet eth0 {ip = "101.0.0.4"}
    ethernet eth0 {ip = "192.168.0.2"}
    resource AncillaryServices
  }
  @Customer @Field
  device EndOfLineMonitor {
    ethernet eth0 {ip = "101.0.0.5"}
    resource MonitorResource
  }
}
```

Listing 6.4: Initial `System` model for the use case example.

Each `Device` also has at least one computing `Resource` and one `ICTInterface`. Compared to the other `Devices`, the only `Device` with two `Resources` is the *DSOComputer*. It has one `Resource` for the *DSCADA* `Function` and one for the *VoltVArController* `Function`. This shows how one device can be split up into different computing resources and how this can be used to group different functionality.

The use case diagram (see Table B.45) also shows the ICT connections. From these, it is possible to define the `ICTInterfaces` of the `Devices`. For each connection to a `Device`, an `ICTInterface` is created. For this use case it is assumed that an IP network is used. Thus, all defined interfaces are of the `ethernet` type and for each `ICTInterface` an IP address is defined. It is also possible to specify other settings in the same manner as the IP address (e.g., `macAddress = "00:0c:a8:..."`).

The `ElectricalComponents` are defined in the same manner as the `ICTComponents`. From the use case diagram (see Table B.45), the main components are identified. Next, `Terminals` are added in order to connect the `ElectricalComponents` with each other. Listing 6.5 shows how the `ExternalSystem`, the MV busbar, the transformer, and the *DERGenerator* are added to the initial `System` model.

```
@@ -33,5 +33,43 @@
   device EndOfLineMonitor {
     ethernet eth0 {ip = "101.0.0.5"}
     resource MonitorResource
   }
+  @Distribution @Process
+  generator ExternalSystem {
+    terminal MVBus
+  }
+  @Distribution @Process
+  busbar MVBus {
+    terminal ExternalSystem
+    terminal MV2LVTransformer
+  }
+  @Distribution @Process
+  transformer MV2LVTransformer {
+    winding MV {
+      terminal mvSide
+    }
+    winding LV {
+      terminal lvSide
+  }}
+  @Distribution @Process
+  busbar LVBus1 {
+    terminal MV2LVTransformer
+    terminal Line1
+  }
+  line Line1 {
+    terminal LVBus1
+    terminal LVBus2
+  }
+  @Distribution @Process
+  busbar LVBus2 {
+    terminal Line1
+    terminal DERGenerator
+    terminal Line2
+  }
+  @DER @Process
+  generator DERGenerator {
+    ethernet eth0 {ip = "192.168.0.1"}
+    terminal LVBus2
+    resource DERResource
+  }
 }
```

Listing 6.5: Adding `ElectricalComponents` to the `System` in Listing 6.4.

As with the `Functions` in Listing 6.1, each `ElectricalComponent` is also placed in a domain and zone. The transformer consists of two `TransformerWindings`: one for the MV side and one for the LV side. Each component also has at least one `Terminal`. These are the electrical connection points of the component. Apart from a `Terminal`, the *DERGenerator* also has a computing `Resource` and an Ethernet `ICTInterface`.

Once all the components have been defined, they must be connected with each other. Only `PhysicalInterfaces` of the same type may be connected. Thus, it is not possible to connect a `Terminal` with an `ICTInterface`. An `ICTInterface` may also only be used in one `Connection`. Consequently, it is not possible for a `PhysicalInterface` to have multiple connections. The only component that does not require `PhysicalInterfaces` is the router. Therefore, it may also have multiple connections. In Listing 6.6, the `Connections` are added to complement the components in Listing 6.4 and Listing 6.5.

```
@@ -73,3 +73,16 @@
```

```
      resource DERResource
    }
+   connect DistributionRTU.eth0 with DSOComputer.eth0
+   connect DistributionRTU.eth1 with StationRouter
+   connect TransformerMonitor.eth0 with StationRouter
+   connect BusMonitor.eth0 with StationRouter
+   connect DERController.eth0 with StationRouter
+   connect DERController.eth1 with DERGenerator.eth0
+   connect EndOfLineMonitor.eth0 with StationRouter
+   connect MVBus.ExternalSystem with ExternalSystem.MVBus
+   connect MV2LVTransformer.MV.mvSide with MVBus.MV2LVTransformer
+   connect LVBus1.MV2LVTransformer with MV2LVTransformer.LV.lvSide
+   connect Line1.LVBus1 with LVBus1.Line1
+   connect LVBus2.Line1 with Line1.LVBus2
+   connect DERGenerator.LVBus2 with LVBus2.DERGenerator
  }
```

Listing 6.6: Adding `Connections` between the components of the `System`.

By connecting the components with each other, the final design step of the `System` model is completed. The previous steps showed how some of the components were added. In Listing 6.7, the complete `System` model is shown, including all components from the use case diagram, see Table B.45. An even more detailed version of the `System` model is also included in Appendix C, which also includes comments.

```
system DistributionSystemVV {
  @Distribution @Operation
  device DSOComputer {
    ethernet eth0 {ip = "10.0.0.1"}
    resource SCADA
    resource VoltVAr
  }
  @Distribution @Station
  device DistributionRTU {
    ethernet eth0 {ip = "10.0.0.2"}
    ethernet eth0 {ip = "101.0.0.1"}
    resource RTUResource
  }
  @Distribution @Station
  router StationRouter
  @Distribution @Field
  device TransformerMonitor {
    ethernet eth0 {ip = "101.0.0.2"}
    resource MonitorResource
  }
  @Distribution @Field
  device BusMonitor {
    ethernet eth0 {ip = "101.0.0.3"}
    resource MonitorResource
  }
  @DER @Field
  device DERController {
    ethernet eth0 {ip = "101.0.0.4"}
    ethernet eth0 {ip = "192.168.0.2"}
    resource AncillaryServices
  }
  @Customer @Field
  device EndOfLineMonitor {
    ethernet eth0 {ip = "101.0.0.5"}
    resource MonitorResource
  }
  @Distribution @Process
  generator ExternalSystem {
    terminal MVBus
  }
  @Distribution @Process
  busbar MVBus {
    terminal ExternalSystem
    terminal MV2LVTransformer
  }
  @Distribution @Process
  transformer MV2LVTransformer {
    winding MV {
      terminal mvSide
    }
    winding LV {
      terminal lvSide
  }}
  @Distribution @Process
  busbar LVBus1 {
    terminal MV2LVTransformer
    terminal Line1
  }
  line Line1 {
    terminal LVBus1
    terminal LVBus2
  }
  @Distribution @Process
  busbar LVBus2 {
    terminal Line1
    terminal DERGenerator
    terminal Line2
  }
  @DER @Process
  generator DERGenerator {
    ethernet eth0 {ip = "192.168.0.1"}
    terminal LVBus2
    resource DERResource
  }
  line Line2 {
    terminal LVBus2
    terminal LVBus3
  }
  @Customer @Process
  busbar LVBus3 {
    terminal Line2
```

```
    terminal Load
  }
  @Customer @Process
  consumer Load {
    terminal LVBus3
  }
  connect DistributionRTU.eth0 with DSOComputer.eth0
  connect DistributionRTU.eth1 with StationRouter
  connect TransformerMonitor.eth0 with StationRouter
  connect BusMonitor.eth0 with StationRouter
  connect DERController.eth0 with StationRouter
  connect DERController.eth1 with DERGenerator.eth0
  connect EndOfLineMonitor.eth0 with StationRouter
  connect MVBus.ExternalSystem with ExternalSystem.MVBus
  connect MV2LVTransformer.MV.mvSide with MVBus.MV2LVTransformer
  connect LVBus1.MV2LVTransformer with MV2LVTransformer.LV.lvSide
  connect Line1.LVBus1 with LVBus1.Line1
  connect LVBus2.Line1 with Line1.LVBus2
  connect DERGenerator.LVBus2 with LVBus2.DERGenerator
  connect Line2.LVBus2 with LVBus2.Line2
  connect LVBus3.Line2 with Line2.LVBus3
  connect Load.LVBus3 with LVBus3.Load
}
```

Listing 6.7: `System` model for the use case example.

Once the `System` model and the `Application` model are defined, the next step is to associate them with each other. This is done by adding a `FunctionMapping` to each `Function` in the `Application`. This means that each `Function` is mapped to a computational `Resource` (i.e., it is defined where the `Function` is executed). In this use case, the mapping is mostly straightforward. Based on the use case diagram in Table B.45, the `Function` declarations are extended with mapping information. Furthermore, the *DERGenerator* `Function` is mapped to the *DERResource*. The mapping information is added to the `Functions` in Listing 6.8.

```
@@ -2,3 +2,3 @@
   @Distribution @Operation
-  function VoltVArController {
+  function VoltVArController at DistributionSystemVV.DSOComputer.VoltVAr {
     consumes Measurements.AggregatedMeasurement gridStatus
@@ -7,3 +7,3 @@
   @Distribution @Station
-  function DSCADA {
+  function DSCADA at DistributionSystemVV.DSOComputer.SCADA {
     emits Measurements.AggregatedMeasurement gridStatus
@@ -14,3 +14,3 @@
   @Distribution @Station
-  function DistributionRTU {
+  function DistributionRTU at DistributionSystemVV.DistributionRTU.RTUResource {
     emits Measurements.AggregatedMeasurement gridStatus
@@ -24,3 +24,3 @@
   @DER @Field
```

```
-  function DERController {
+  function DERController at DistributionSystemVV.DERController.AncillaryServices {
     provides DERCtrlInterfaces.DERVoltVArCurve voltVar
@@ -31,3 +31,3 @@
   @DER @Process
-  function DERGenerator {
+  function DERGenerator at DistributionSystemVV.DERGenerator.DERResource {
     provides DERCtrlInterfaces.DERDirectControls directControls
@@ -36,12 +36,12 @@
   @Distribution @Field
-  function TransformerMonitor {
+  function TransformerMonitor at ⤶
    ↳ DistributionSystemVV.TransformerMonitor.MonitorResource {
      provides Measurements.GridMeasurements measurements
   }
   @Distribution @Field
-  function BusMonitor {
+  function BusMonitor at DistributionSystemVV.BusMonitor.MonitorResource {
     provides Measurements.GridMeasurements measurements
   }
   @Customer Field
-  function EndOfLineMonitor {
+  function EndOfLineMonitor at DistributionSystemVV.EndOfLineMonitor.MonitorResource {
     provides Measurements.GridMeasurements measurements
   }
```

Listing 6.8: The `Functions` from Listing 6.9 mapped to the `Resources` in Listing 6.7.

The `Application` model is completed by adding the `FunctionMappings` to the `Functions`. This model is illustrated in Listing 6.9.

```
application VoltVArControlCoordinated {
  @Distribution @Operation
  function VoltVArController at DistributionSystemVV.DSOComputer.VoltVAr {
    consumes Measurements.AggregatedMeasurement gridStatus
    requests DERCtrlInterfaces.DERVoltVArCurve dscadaVoltVArCurve
  }
  @Distribution @Station
  function DSCADA at DistributionSystemVV.DSOComputer.SCADA {
    emits Measurements.AggregatedMeasurement gridStatus
    provides DERCtrlInterfaces.DERVoltVArCurve voltVArCurve
    consumes FieldInformation.GridStatus gridStatusRTU
    requests DERCtrlInterfaces.DERVoltVArCurve rtuVoltVArCurve
  }
  @Distribution @Station
  function DistributionRTU at DistributionSystemVV.DistributionRTU.RTUResource {
    emits Measurements.AggregatedMeasurement gridStatus
    provides DERCtrlInterfaces.DERVoltVArCurve voltVArCurve
    requests DERCtrlInterfaces.DERVoltVArCurve derVoltVArCurve
    requests Measurements.GridMeasurements derMeasurements
    requests Measurements.GridMeasurements transformerMeasurements
    requests Measurements.GridMeasurements busMeasurements
    requests Measurements.GridMeasurements eolMeasurements
  }
  @DER @Field
  function DERController at DistributionSystemVV.DERController.AncillaryServices {
    provides DERCtrlInterfaces.DERVoltVArCurve voltVar
    provides Measurements.GridMeasurements measurements
    requests DERCtrlInterfaces.DERDirectControls derDirectControls
    requests Measurements.GridMeasurements derMeasurements
  }
  @DER @Process
  function DERGenerator at DistributionSystemVV.DERGenerator.DERResource {
    provides DERCtrlInterfaces.DERDirectControls directControls
    provides Measurements.GridMeasurements measurements
  }
  @Distribution @Field
  function TransformerMonitor at ⤶
         ↳ DistributionSystemVV.TransformerMonitor.MonitorResource {
    provides Measurements.GridMeasurements measurements
  }
  @Distribution @Field
  function BusMonitor at DistributionSystemVV.BusMonitor.MonitorResource {
    provides Measurements.GridMeasurements measurements
  }
  @Customer Field
  function EndOfLineMonitor at DistributionSystemVV.EndOfLineMonitor.MonitorResource {
    provides Measurements.GridMeasurements measurements
  }
  module Measurements {
    interface GridMeasurement {
        readonly attribute float32 voltage
    }
    eventtype AggregatedMeasurement {
      public float32 vTM, vBM, vEOLM, vDER
  }}
  module DERCtrlInterfaces {
    interface DERVoltVArCurve {
      attribute float32 v1, v2, v3, v4
      attribute float32 q1, q2, q3, q4
    }
    interface DERDirectControls {
      attribute float32 qSetPoint
  }}
  connect DERController.derDirectControls with DERGenerator.directControls
  connect DERController.derMeasurements with DERGenerator.measurements
  connect DistributionRTU.derVoltVArCurve with DERController.voltVar
  connect DistributionRTU.derMeasurements with DERController.measurements
  connect DistributionRTU.transformerMeasurements with TransformerMonitor.measurements
  connect DistributionRTU.busMeasurements with BusMonitor.measurements
  connect DistributionRTU.eolMeasurements with EndOfLineMonitor.measurements
  connect DSCADA.gridStatusRTU with DistributionRTU.gridStatus
  connect DSCADA.rtuVoltVArCurve with DistributionRTU.voltVArCurve
  connect VoltVArController.gridStatus with DSCADA.gridStatus
  connect VoltVArController.dscadaVoltVArCurve with DSCADA.voltVArCurve
}
```

Listing 6.9: Business case and functional specification for the example use case.

6.2.3 Information and Communication Layers Design

Once an initial design of the `Application` and the `System` models is available, it is time to start with a detailed modeling. In SGAM, the goal of the information layer is to describe the information flow between the different components of the tackled use case and which data model is used (e.g., IEC 61850, CIM) [25]. This information also needs to be provided in the rapid engineering methodology . Step three of the *specification* phase is to provide a detailed data model specification of the information exchanged in the use case. As seen from Listing 6.9, an initial definition of the information layer (i.e., the definition of `Interfaces` and `Events`) is done in the design of the business and function layers (see Section 6.2.1). This is also needed in order to provide a general picture of the interaction between `Functions`. However, the defined information is only using a generic data model.

In order to use other data models, a protocol mapping is used. Section 4.4.3 describes how this is done for IEC 61850. For this use case, IEC 61850 will be used as communication protocol for the information exchange between the field, the station, and the operation zones. As depicted in Figure 4.14c, the *MMXU* `Interface` is declared as `abstract`. This means that it cannot be directly implemented by a `ServiceImplementation`. Instead, an abstract `Interface` can only be implemented through inheritance by a non-abstract `Interface`. Both `Interfaces` and `Events` can inherit, which means that all attributes, operations, and/or state members defined in the parent are also inherited by the child.

As already described in Section 5.3, the possibility for `Interfaces` and `Events` to inherit also provides a solution for how different data models and protocols can be used by `ServiceImplementations`. If the *MMXU* `Interface` from Figure 4.14c is used as a parent for the *GridMeasurement* `Interface` from Listing 6.9 all `ServiceImplementations` that now implement *GridMeasurement* also have access to the attributes defined by the *MMXU* `Interface`. Using the same approach, the *DERVoltVArCurve* and the *DERDirectControls* `Interfaces` in Listing 6.9 can also be changed into IEC 61850 `Interfaces`. In this case, *DERVoltVArCurve* inherits from the *FMAR* LN, which is used to define a volt-VAr droop curve and is defined in the IEC 61850-90-7 technical report [71]. The *DERDirectControls* inherits from the *DRCC* LN. It is used for DER supervisory control and was first defined in the IEC 61850-7-420 part [65]. Later the *DRCC* LN was adapted in the IEC 61850-90-7 technical report [71]. The changes made to the interfaces are shown in Listing 6.10.

```
@@ -47,5 +47,3 @@
   module Measurements {
-    interface GridMeasurement {
-      readonly attribute float32 voltage
-    }
+    interface GridMeasurement : IEC61850.MMXU { }
     eventtype AggregatedMeasurement {
@@ -54,9 +52,5 @@
   module DERCtrlInterfaces {
-    interface DERVoltVArCurve {
-      attribute float32 v1, v2, v3, v4
-       attribute float32 q1, q2, q3, q4
-     }
+     interface DERVoltVArCurve : IEC61850.FMAR { }
-     interface DERDirectControls {
-       attribute float32 qSetPoint
-   }}
+     interface DERDirectControls : IEC61850.DRCC { }
+   }
    connect DERController.derDirectControls with DERGenerator.directControls
```

Listing 6.10: Inheritance from IEC 61850 `Interfaces` is added to the `Interfaces` from Listing 6.9.

For the `Interfaces` that inherit, a clean up of the `Attributes` may also be needed. The *MMXU* `Interface` already has an `Attribute` for voltage measurements (actually multiple `Attributes` are available but in this case the arithmetic average of the phase to phase voltage, *AvPPVPhs*, would be the most fitting). If the original *voltage* attribute is not removedm *GridMeasurement* will contain two `Attributes` used for the same measurement. This may lead to ambiguousness and should therefore be avoided. The same applies to the inheritance of the *FMAR* and the *DRCC* `Interfaces`. Therefore, the old `Attributes` of *DERVoltVArCurve* and *DERDirectControls* are also removed.

Apart from the data models, different protocols, working on different Open Systems Interconnection (OSI) layers, can be used. In PSAL, the lower OSI layers (layers 1-4) can be defined for each `ICTInterface` and for ICT `Connections`. For the upper OSI layers (layers 5-7), definitions or configurations are specified, if they are needed, in the `ServiceImplementations`.

6.3 Applying Phase II: Implementation

This section shows the application of the *implementation* phase. The different steps of this phase are shown in Figure 4.15, and as steps 4–6 in Figure 5.6. For this use case, focus is put on the implementation of the SubApps[499] *VoltVArController* and *DERController*. However, non of the SubApps[499] are automatically generated, since this would require an existing set of implementations. Instead both are implemented manually.

Once the `Application` and the `System` have been specified and the correct data model has been defined for the `Interfaces` and `Events`, a transformation is made into the chosen programming technique. For this work, it means a transformation to IEC 61499 using the rules defined in Section 4.4.2. After the transformation, IEC 61499 is used to define the behavior of each `Function`.

First of all, the `Application` and `System` are transformed into equivalent IEC 61499 Application[499] and system models. For this, the mapping from Figure 4.10 is used. From the `Application` a corresponding *VoltVArControlCoordinated* Application[499] is created, which is shown in Figure 6.2. The second step is to generate a SubAppType[499] for each `Function` in Listing 6.9. Each SubAppType[499] will have Plugs[499] and Sockets[499] representing the

`ServiceImplementations` of the original `Functions`. Each Socket[499]/Plug[499] is defined by an AdapterType[499]. The AdapterType[499] is transformed from the `Interface` or `Event` that is implemented by the `ServiceImplementation`. This transformation uses the mappings in Figure 4.11 and Figure 4.12. In the third step, the SubApp[499] instances of the SubAppTypes[499] are added to the Application[499]. Based on the `Connections` in the `Application` in Listing 6.9 the Sockets[499] and Plugs[499] are connected with each other through adapter connections.

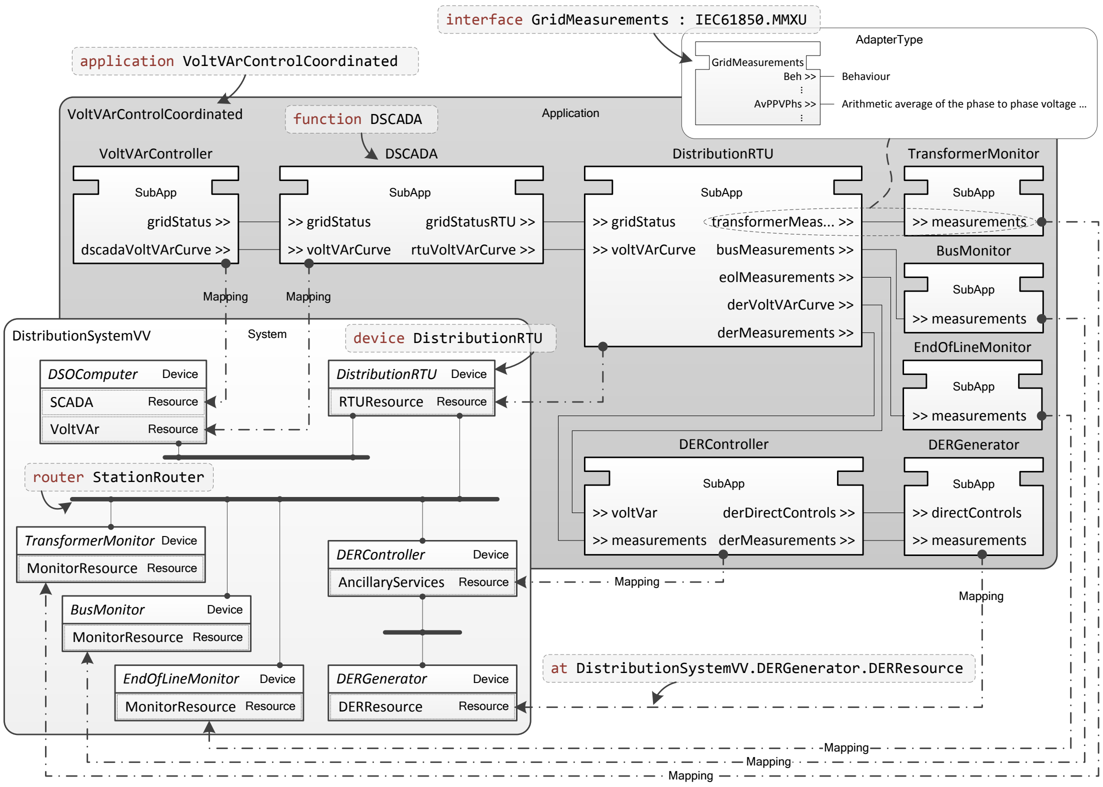


Figure 6.2: *VoltVArControlCentralized* as IEC 61499 Application[499].

Figure 6.2 also shows how the `System` from Listing 6.7 is transformed into a corresponding IEC 61499 system model with Devices[499] and Resources[499]. This transformation only considers the `ICTComponents` of the `System` model, as well as any `ElectricalComponents` that contain computing `Resources`. Finally, the figure also illustrates how Mappings[499] are generated from the `FunctionMappings` in Listing 6.9. Each SubApp[499] is mapped to its respective Resources[499] (e.g., the *DSCADA* SubApp[499] is mapped to the *SCADA* Resource[499] in the *DSOComputer* Device[499]).

Figure 6.2 shows how transformations between PSAL `Functions` and IEC 61499 SubApps[499] are made. However, the resulting SubApps[499] still do not contain any content. As was already stated in Section 6.1, it is assumed that for this use case only the *VoltVAr-Controller* and the *DERController* need to be implemented. Their implementations will be handled in the following sections. For the other `Functions`, it is assumed that they are already implemented and installed in the field. This means that the SubAppTypes[499] representing these already existing `Functions` do not need to be implemented.

6.3.1 Implementing the DER Control Functionalities

The implementation of the DER control functionalities needs to be done by the DER manufacturer. In this case, it is assumed that the DER manufacturer also uses the same rapid engineering methodology as the DSO. This makes the specification of the ordered functionality much simpler since the `Application` and `System` specifications made by the DSO can be used directly. Thus, when the DSO orders the new control functionalities it also hands over the PSAL specifications to the DER manufacturer . These are then used directly by the DER manufacturer to further implement the ordered functionality.

The ordered functionality is a volt-VAr control (also called Q(U) control) for the inverter of the DER. This is a very simple control, where the reactive power output of the DER depends on the measured voltage at the point of coupling. Usually this relationship between reactive power and voltage follows a so called droop curve. An example of a droop curve for volt-VAr control was already shown in Figure 3.3.

Normally, when new functionality is implemented in an inverter, a firmware update is needed. In this case however, the DER inverter uses the framework developed in the OpenNES project [118]. Specially focusing on inverters, it provides a flexible software architecture that allows deployment of new functionality on demand. Furthermore, the OpenNES ICT and automation solution were developed to be a seamless extension of the rapid prototyping method in this work. Figure 6.3 provides a brief overview of the main project idea.

In order to remotely program DER controller functions or IEDs, an appropriate software architecture is needed on these components. For this purpose, the SmartOS has been conceptualized within the OpenNES project. It provides a flexible component-based architecture which allows remote programmability as well as comprehensive configuration possibilities. In Figure 6.3, the SmartOS is visualized as part of the DER. It contains four main parts: *(i)* pluggable software components, *(ii)* a Virtual Functional Bus (VFB) for communication between components, *(iii)* basic security functionality, and *(iv)* connectivity to external devices [118].

Especially the software components are of interest for this thesis. Software components are pluggable modules, which can be dynamically removed or added within the OpenNES architecture. They can either be developed and delivered by the DER manufacturer or developed by an external certified partner (e.g., plant operator, system integrator). Different types of components are available. Some have fixed functionality (e.g., basic inverter functionality). Others can be programmed from an external IEC 61499 engineering environment. This means that the component can be remotely programmed or updated during runtime [118].

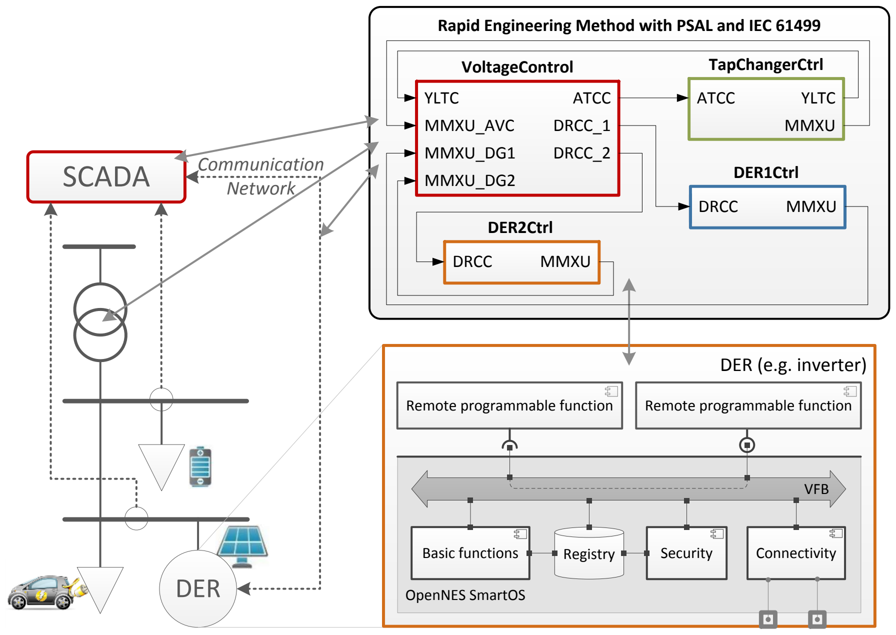


Figure 6.3: Overview of the OpenNES system architecture concept, adapted from [118].

The programmable components can be seamlessly used together with the rapid engineering method in this work. Since they are programmable using IEC 61499 they can be directly comparable to the SubApps[499] and SubAppTypes[499] generated from the `Functions`. For this use case it means that the DER manufacturer implements the *DERController* SubAppType[499] and deploys it as a software component to the OpenNES DER.

For the implementation of the *DERController* SubAppType[499] the interactions with the other SubApps[499] are studied. These are found in Appendix B.4 as well as in Table 6.1, where the steps involving the *DERController* are summarized.

Table 6.1: Overview of the steps with the *DER Controller* from Scenario 6 and Scenario 7.

Scenario 6					
Scenario Name		Distribution system voltage control			
Step No.	**Event**	**Description of Process**	**Information Producer**	**Information Receiver**	**Information Exchanged**
4	Time trigger	The Distribution RTU requests voltage measurement from DER Controller.	Distribution RTU	DER Controller	IEX-1
8		The DER Controller returns voltage measurement to Distribution RTU.	DER Controller	Distribution RTU	IEX-2
14		The Distribution RTU forwards the requests.	Distribution RTU	DER Controller	IEX-4

15		The DER Controller returns its current curve points.	DER Controller	Distribution RTU	IEX-5
19		The new curve setpoints are forwarded.	Distribution RTU	DER Controller	IEX-6
22	Receives curve points	The DER Controller uses the new volt-VAr curve points			
Scenario 7					
Scenario Name	DER volt-VAr control				
Step No.	**Event**	**Description of Process**	**Information Producer**	**Information Receiver**	**Information Exchanged**
1	Time trigger	The DER Controller requests current voltage level from the DER Generator.	DER Controller	DER Generator	IEX-1
2		The DER Generator returns voltage measurement to DER Controller.	DER Generator	DER Controller	IEX-2
3		The DER Controller calculates a new reactive power setpoint based on its current volt-VAr curve.			
4		The new reactive power setpoint is sent to the DER Generator	DER Controller	DER Generator	IEX-7

Apart from handling communication requests of different attributes (voltage measurement and curve points), the main work is represented by Step 22 of Scenario 6 and Step 3 of Scenario 7 in Table 6.1. Based on a volt-VAr curve, represented by a number of voltage and reactive power set points, the *DERController* calculates a new reactive power set point for the *DERGenerator*. Furthermore, the set points of the volt-VAr curve should be changeable through the *voltVAr* interface to the *DistributionRTU* (see Figure 6.2).

The implementation of the *DERController* SubAppType[499] is seen in Figure 6.4. Left and right in the figure the Sockets[499] and Plugs[499] of the SubAppType[499] are shown as adapter FBs[499] (i.e., the *voltVar*, *measurements*, *derDirectControls*, and *derMeasurements* FBs[499]). These are representations of the interfaces of the SubAppType[499]. This means that if data is written on the *voltVar* Socket[499] of the *DERController* SubApp[499] in Figure 6.2, it arrives on the output side of the *voltVar* adapter FB[499] in Figure 6.4.

The *CurvePointBuffer* FB[499] is a buffer where the current volt-VAr curve points are stored. In this use case four points are used to represent the droop curve shown in Figure 3.3. The points can be read and written by anyone connected to the *voltVar* Socket[499] of the *DERController* SubApp[499]. The current curve points are also transfered to the *DroopControl* FB[499] as an array. The main cycle of the SubAppType[499] is handled by the *MainCycle* FB[499]. It triggers an output event every 500 ms. First a new voltage measurement is retrieved from the *derMeasurements* interface. After that the voltage measurement is forwarded with an event to the *DroopControl* FB[499]. It calculates

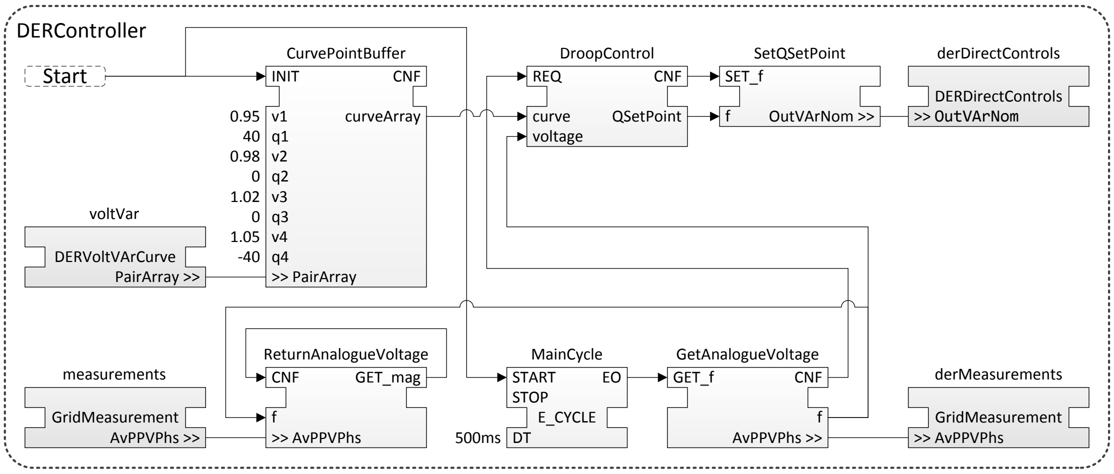


Figure 6.4: Implementation of the *DERController* SubAppType[499].

a new reactive power set point and sends this on the *derDirectControls* interface to the *DERGenerator*. Moreover, the voltage measurement from the *DERGenerator* is also buffered in the *ReturnAnalogueVoltage* FB[499], where it can be read through the *measurements* interface. All FBs[499] that do not have a type name (e.g., *CurvePointBuffer*) are also implemented as SubApps[499]. They are mainly used to handle and access data and their specific implementation is not important for this use case.

6.3.2 Implementing the DSO Control Functionalities

The implementation of the DSO control functionalities follows a similar pattern as in Section 6.3.1. In fact, the DSO may implement this functionality itself but may also be commission it to a third party (e.g., a system integrator). For both cases, the implementation mainly consists of implementing the *VoltVArController* SubAppType[499]. As with the *DERController*, the implementation is based on the use case steps in Table B.50. Table 6.3 summarizes the steps involving the *VoltVArController*.

Table 6.3: Steps involving the *Volt-VAr Controller* for Scenario 6.

Scenario 6					
Scenario Name		Distribution system voltage control			
Step No.	**Event**	**Description of Process**	**Information Producer**	**Information Receiver**	**Information Exchanged**
10		The D-SCADA forwards measurements to the Volt-VAr Controller.	D-SCADA	Volt-VAr Controller	IEX-3
11		The Volt-VAr Controller calculates the voltage spread.			

12	Voltage spread above limit	The Volt-VAr Controller requests the current volt-VAr curve points from the DER Controller.	Volt-VAr Controller	D-SCADA	IEX-4
17		The D-SCADA forwards the current curve points.	D-SCADA	Volt-VAr Controller	IEX-5
18		The Volt-VAr Controller calculates new curve setpoints for the DER Controller			
19		The new curve setpoints are sent to the DER Controller.	Volt-VAr Controller	D-SCADA	IEX-6

From the steps, it is clear that the *VoltVArController* does not need an own cycle, as compared to the *DERController*. The functionality of the *VoltVArController* is triggered when new measurements are received from the *DSCADA*. This causes the *VoltVArController* to calculate the voltage spread and only if the spread is too high it will calculate new curve points for the *DERController*.

In Figure 6.5, the implementation of the *VoltVArController* SubAppType[499] from Figure 6.2 is shown. Two adapter Plugs[499] are implemented by the *VoltVArController*. The *gridStatus* Plug[499] subscribes the important measurements from the grid, and the *dscadaVoltVArCurve* Plug[499] requests the control interfaces from the field devices (i.e., from the *DERController*). Using the grid measurements, the voltage spread (i.e., the difference between highest and lowest voltage measurement) in the grid is calculated. In case of a too high spread, new voltage set points for the field devices are calculated. For the *DERController*, this means that a new volt-VAr droop curve is calculated.

A new volt-VAr droop curve is easily calculated by changing the dead band of the curve. This means that if the voltage spread is too high the dead band is decreased [129]. Left and right in Figure 6.5 the Plugs[499] of *VoltVArController* are shown (i.e., the *gridStatus* and *fieldControls* FBs[499]). The *VoltVArController* SubAppType[499] also contains two other SubApps[499]: *ActivateDroop* and *CurvePoints*. These SubApps[499] are only used to access the important data for this use case. The *CurvePoints* SubApp[499] provides the points of the droop curve as an array, with pairs of volt-VAr values (i.e., *pointArray* for reading and *pointArrayW* for writing).

Every time a new *EMIT* event is emitted (i.e., new measurements are published by *DSCADA*), the voltage spread is calculated from the new measurements. Thereafter, the current droop curve points are requested from the DER. Once these arrive they are forwarded, together with the voltage spread, to the *ChangeDeadband* FB[499]. If the spread is higher than the allowed *Threshold* the dead band of the droop curve is decreased by the *Change* amount. Next, the *ChangeDeadband* FB[499] sends the new droop curve values to the *DERController*.

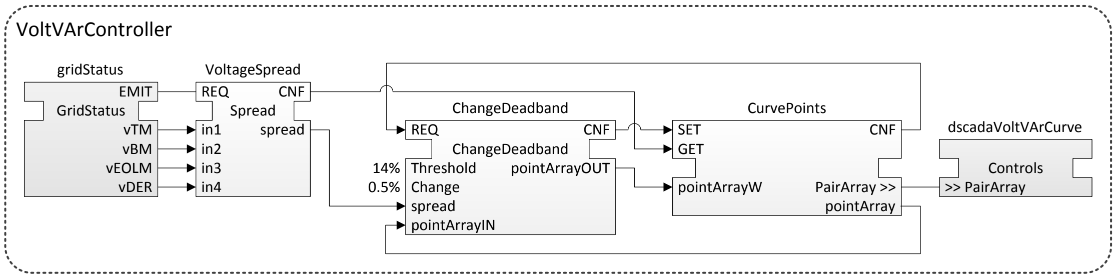


Figure 6.5: Implementation of the *VoltVArController* SubApp[499] from Figure 6.2.

6.4 Applying Phase III–IV: Validation and Deployment

Once the implementation of the different SubApps[499] is finished, the next step is the *validation and deployment* phase. For this use case, two validations are made: the implementation of the *DERController* is validated using a simulation and the control functions associated with the DSO are validated in a laboratory environment.

6.4.1 Validating the DER Control Implementation

The validation of the DER control implementation for this use case is intended to be done by the DER manufacturer. The goal is to test the volt-VAr control of the *DERController* using a simulative validation. It is not intended to test how the curve set points can be adapted, and also not how voltage measurements are read by the *DSCADA*. Using these assumptions, the validation case can be simplified to only concentrate of the *DERController* and the *DERGenerator* `Functions`.

Scenario and Test Case Description

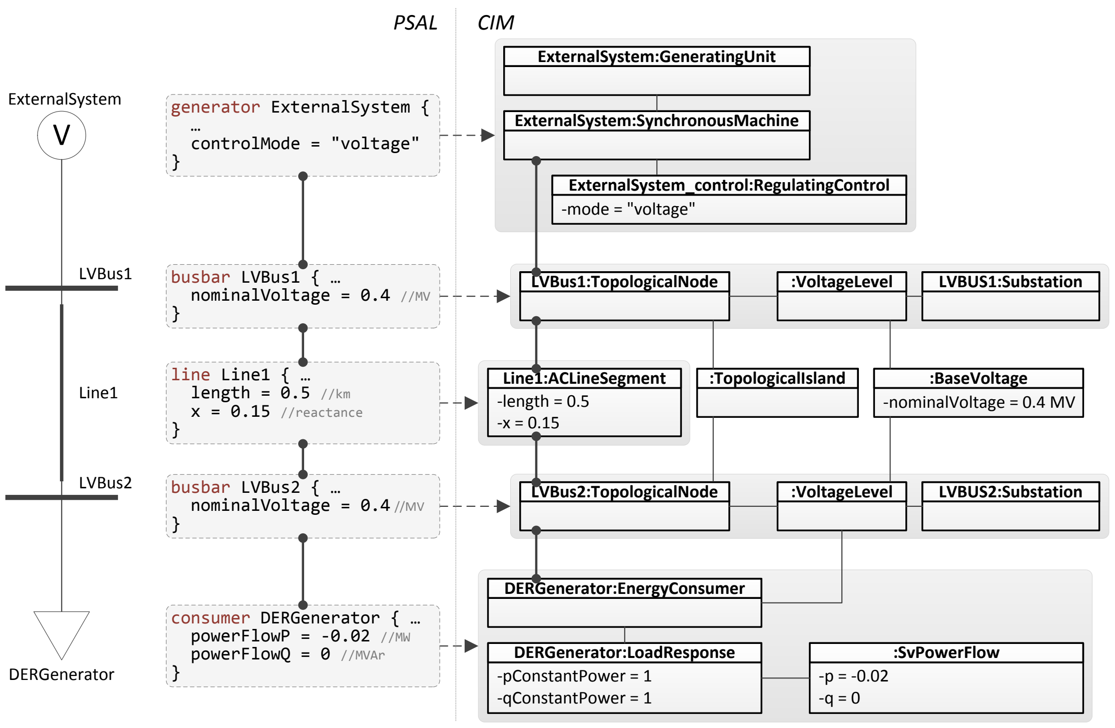
If only the *DERController* and the *DERGenerator* `Functions` are studied the `Application` from Listing 6.9 and `System` from Listing 6.7 can be simplified. Furthermore, the complete model of the power system is also not needed to only test the volt-VAr control of the *DERController*. A substitute power system model is used containing only a slack generator, the *DERGenerator*, and a line connecting their busbars. The resulting simplified `Application` and `System` are seen in Listing 6.11. The single-line diagram of the power system model is also seen in Figure 6.6.

```
application VoltVArControlTest {
  function DERController at VoltVArSystemTest.DERController.AncillaryServices {
    requests DERCtrlInterfaces.DERDirectControls derDirectControls
    requests Measurements.GridMeasurements derMeasurements
  }
  function DERGenerator at VoltVArSystemTest.DERGenerator.DERResource {
    provides DERCtrlInterfaces.DERDirectControls directControls
    provides Measurements.GridMeasurements measurements
  }
  module Measurements {
    interface GridMeasurement : IEC61850.MMXU { }
  }
  module DERCtrlInterfaces {
    interface DERDirectControls : IEC61850.DRCC { }
  }
  connect DERController.derDirectControls with DERGenerator.directControls
  connect DERController.derMeasurements with DERGenerator.measurements
}
system VoltVArSystemTest {
  device DERController {
    ethernet eth0 {ip = "192.168.0.2"}
    resource AncillaryServices
  }
  generator ExternalSystem {
    terminal LVBus
    controlMode = "voltage"
  }
  busbar LVBus1 {
    terminal ExternalSystem
    terminal Line1
    nominalVoltage = 0.4 //MV
  }
  line Line1 {
    terminal LVBus1
    terminal LVBus2
    length = 0.5 //km
    x = 0.15 //reactance
  }
  busbar LVBus2 {
    terminal Line1
    terminal DERGenerator
    nominalVoltage = 0.4 //MV
  }
  consumer DERGenerator {
    ethernet eth0 {ip = "192.168.0.1"}
    terminal LVBus2
    resource DERResource
    powerFlowP = -0.02 //MW
    powerFlowQ = 0 //MVAr
  }
  connect DERController.eth1 with DERGenerator.eth0
  connect LVBus1.ExternalSystem with ExternalSystem.LVBus
  connect Line1.LVBus1 with LVBus1.Line1
  connect LVBus2.Line1 with Line1.LVBus2
  connect DERGenerator.LVBus2 with LVBus2.DERGenerator
}
```

Listing 6.11: Simplified `Application` and `System` model for the validation of the *DERController*.

Compared to the original `Application` in Listing 6.9 and the original `System` in Listing 6.7, most of the changes are simple deletions of the parts that are not needed for this validation. One of the significant changes is that the type of the *DERGenerator* is changed from `generator` to `consumer`. The reason for this is that the simplest way to represent DERs in PowerFactory is to use a negative load (i.e., a negative `consumer`). Apart from this, parameters are added to the `ElectricalComponents` as `IDValuePairs` (e.g., `controlMode = "voltage"`)). Each of these parameters is transformed into a corresponding parameter in CIM, and they are needed for the power system simulation.

Using the `System` model in Listing 6.11, a CIM model can be generated from the `ElectricalComponents` and their `Connections`. Figure 6.6 shows how this generation is made. Each `ElectricalComponent` is transformed into one or more CIM objects. Furthermore the connections between the components in Figure 6.6 represents the `Terminals` and the `Connections` between them. This transformation is specifically defined in order to be compliant with PowerFactory. Thus if another simulation tool is used other/further parameters may be needed for the `ElectricalComponents`.



```
generator ExternalSystem {
  …
  controlMode = "voltage"
}
```

```
busbar LVBus1 { …
  nominalVoltage = 0.4 //MV
}
```

```
line Line1 { …
  length = 0.5 //km
  x = 0.15 //reactance
}
```

```
busbar LVBus2 { …
  nominalVoltage = 0.4 //MV
}
```

```
consumer DERGenerator { …
  powerFlowP = -0.02 //MW
  powerFlowQ = 0 //MVAr
}
```

Figure 6.6: Generation of CIM model from the PSAL `System` model.

Validation Environment and Concept

The generated CIM model is imported into PowerFactory. On the left side of Figure 6.7 the imported model is seen. It shows the two busbars with the slack generator *ExternalSystem* and the DER. Using only PowerFactory load flow and dynamic analysis can be performed on the model. However, unless manually modeled in PowerFactory, such simulations do not include the control behavior of the *DERController*. As already mentioned in

Section 4.6.1, the rapid engineering methodology solves this by means of co-simulation between simulation tools [150, 143, 140]. The idea is that the control functionality is executed in 4diac-rte and the power system model is simulated in PowerFactory.

In order to connect PowerFactory with 4diac-rte in a co-simulation setup, different approaches are possible. One of the most convenient ways is to include an external Dynamic Link Library (DLL) into PowerFactory that handles the connection with 4diac-rte. During dynamic simulations it is possible to call user defined functions in the DLL. This is done through a DIgSILENT Simulation Language (DSL) model, which is executed each step of the simulation [144]. Since the external DLL can contain any user functionality it is possible to use it for communication with 4diac-rte. For this validation case, a TCP/IP connection is used, where the external DLL connects to a TCP server in 4diac-rte. Figure 6.7 shows the simulation setup between PowerFactory and 4diac-rte.

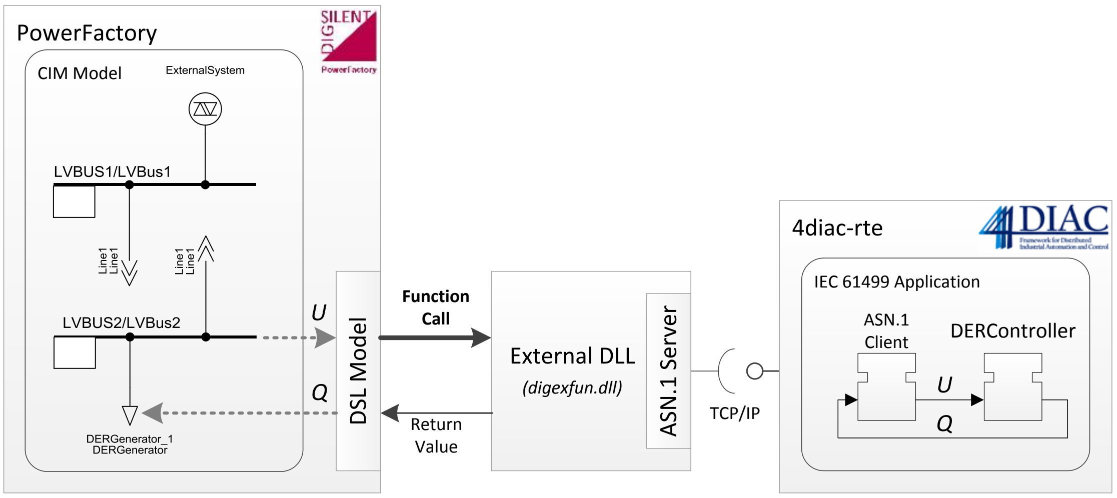


Figure 6.7: Co-simulation setup between PowerFactory and 4diac-rte, including the imported CIM model in PowerFactory.

As seen in Figure 6.7, the *DERGenerator* is simulated within PowerFactory. This means that the communication between the *DERController* and the *DERGenerator* is now realized with the communication between PowerFactory and 4diac-rte. From the figure, it is seen that ASN.1 is used for the communication between PowerFactory and 4diac-rte. The reason for this is that PowerFactory does not support IEC 61850. For such cases, the easiest way use ASN.1 is to override the use of IEC 61850. To achieve this, the `ServiceImplementations` in Listing 6.11 must be changed in order use ASN.1 instead of IEC 61850. These changes are seen in Listing 6.12 where `ProtocolUsages` are added. As a result, the IEC 61850 information model is still used (i.e., the naming of the `Attributes` stays the same), but the data format for the communication is using ASN.1. With these additions to the `ServiceImplementations`, ASN.1 is used for the generation of communication SIFBs. Thus, instead of an IEC 61850 ID configuration string as is seen in Figure 5.5, an ASN.1 configuration string is generated.

```
@@ -2,8 +2,8 @@
   function DERController at VoltVArSystemTest.DERController.AncillaryServices {
-    requests DERCtrlInterfaces.DERDirectControls derDirectControls
+    requests DERCtrlInterfaces.DERDirectControls derDirectControls using StdProt.ASN1
-    requests Measurements.GridMeasurements derMeasurements
+    requests Measurements.GridMeasurements derMeasurements using StdProt.ASN1
   }
   function DERGenerator at VoltVArSystemTest.DERGenerator.DERResource {
-    provides DERCtrlInterfaces.DERDirectControls directControls
+    provides DERCtrlInterfaces.DERDirectControls directControls using StdProt.ASN1
-    provides Measurements.GridMeasurements measurements
+    provides Measurements.GridMeasurements measurements using StdProt.ASN1
   }
```

Listing 6.12: Inheritance from IEC 61850 `Interfaces` is added to the `Interfaces` from Listing 6.9.

Performed Experiment and Results

Once the SIFBs are generated the validation can be started. To start the co-simulation 4diac-rte must be started first (i.e., the *DERController* is deployed). Thereafter, the simulation in PowerFactory is started, whereupon PowerFactory connects to 4diac-rte through the external DLL. In each simulation step values are exchanged between PowerFactory and 4diac-rte. For this setup to work, the simulation in PowerFactory must be run in real-time (i.e., the simulation time is synchronized with the computer clock). The voltage at *LVBus2* in Figure 6.7 is measured and sent through the DLL to 4diac-rte. Every time a new voltage measurement is received a new reactive power set point is calculated by the *DERController* SubApp[499]. The new set point is sent back to the simulation in PowerFactory where it is applied to the *DERGenerator* load. The connection between PowerFactory and 4diac-rte is asynchronous, which means that PowerFactory does not wait for a return value from 4diac-rte before it continues with the simulation of the next step. Since both PowerFactory and 4diac-rte run in real-time this setup mimics the behavior of a real system.

In Figure 6.8, the results of the co-simulation are shown. For the simulation, the voltage was varied as a step function, as seen in the upper graph. The lower graph shows the response of the *DERController*. As seen the simulated reactive power follows the expected volt-VAr curve (i.e., the Q(U) curve). From this figure the dead-band and the saturations are clearly seen. Although the voltage changes between 10 s and 14 s the reactive power output does not change.

This simulation shows the validation of the *DroopControl* FB[499] of the *DERController* (see Figure 6.4). Since this is only one part more tests would normally be needed in order to completely validate the implementation of the *DERController*. For this thesis however, it is assumed that they have already been completed. Therefore the next step is to validate the DSO control functions and thus also the whole use case. This is described in the next section.

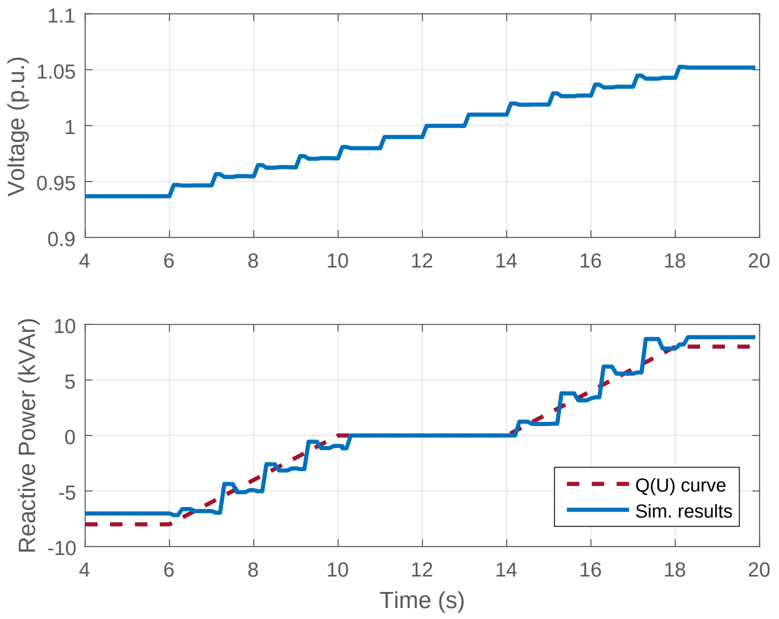


Figure 6.8: Validation results for the co-simulation of the *DERController*.

6.4.2 Validating the DSO Control Functions

Once the *DERController* has been implemented and validated it is also possible to validate the DSO functions. As already explained in Section 6.1, it is assumed that the functions of the *DSCADA* and the *DistributionRTU* are already existing. Thus the remaining functionality to validate is the *VoltVArController*. This validation is made in a laboratory environment, and can be done either directly by the DSO (i.e., a utility operator), but also by a system integrator.

Scenario and Test Case Description

For the test case, the following scenario is used. A voltage change in the grid causes the voltage spread to increase above its allowed threshold. This causes the control algorithm in *VoltVArController* to calculate a new volt-VAr droop curve for the *DERController*. With the new curve the *DERController* will be more sensitive to voltage deviations. This in turn will decrease the voltage spread.

Compared to the validation in Section 6.4.1, this test case shows how the rapid engineering method is used to deploy an application to real devices. Furthermore, it also includes communication between multiple devices using IEC 61850, which needs to be configured. Finally, the goal is also to validate the implementation of the *VoltVArController* SubApp[499] shown in Figure 6.5.

For this scenario, the `Application` from Listing 6.9 and the `System` from Listing 6.7 are used. Since no simulation is used the `ElectricalComponents` are not directly used, although they represent the grid emulated in the laboratory. From the `ICTComponents` the *TransformerMonitor*, the *BusMonitor*, and the *EndOfLineMonitor* are not available in the laboratory. Thus their `Functions` will instead by assigned to a new `Resource` in the *DistributionRTU*. These changes are shown in Listing 6.13 and Listing 6.14.

```
@@ -9,5 +9,6 @@
   device DistributionRTU {
     ethernet eth0 {ip = "10.0.0.2"}
     ethernet eth0 {ip = "101.0.0.1"}
     resource RTUResource
+    resource MonitorResource
   }
```

Listing 6.13: Adding a new `Resource` to the *DistributionRTU* in Listing 6.7.

```
@@ -36,11 +36,11 @@
   @Distribution @Field
-  function TransformerMonitor at ⤶
     ↳ DistributionSystemVV.TransformerMonitor.MonitorResource {
+  function TransformerMonitor at ⤶
     ↳ DistributionSystemVV.DistributionRTU.MonitorResource {
     provides Measurements.GridMeasurements measurements
   }
   @Distribution @Field
-  function BusMonitor at DistributionSystemVV.BusMonitor.MonitorResource {
+  function BusMonitor at DistributionSystemVV.DistributionRTU.MonitorResource {
     provides Measurements.GridMeasurements measurements
   }
   @Customer Field
-  function EndOfLineMonitor at DistributionSystemVV.EndOfLineMonitor.MonitorResource {
+  function EndOfLineMonitor at DistributionSystemVV.DistributionRTU.MonitorResource {
     provides Measurements.GridMeasurements measurements
```

Listing 6.14: Mapping is changed for the monitor `Functions` in Listing 6.9.

Validation Environment and Concept

The laboratory setup is illustrated in Figure 6.9. As already discussed above, the number of hardware ICT components are reduced. Figure 6.9 also shows to which Device[499] the different SubApps[499] are mapped. The *VoltVArController* and the *DSCADA* SubApps[499] are mapped to the *DSOComputer* Device[499]. The *DistributionRTU* and the monitor SubApps[499] are mapped to the *DistributionRTU* Device[499] and the *DERController* SubApp[499] is mapped to the *DERController* Device[499].

The controller hardware used in the laboratory consist of a normal laptop with Windows 7 for the *DSOComputer* and two Raspberry Pis (Raspberry Pi 1 Model B+) with Raspbian are used for the *DistributionRTU* and *DERController*. 4diac-rte is available for multiple platforms, including Windows and Raspbian, and thus there was no problem of installing it to the three Devices[499].

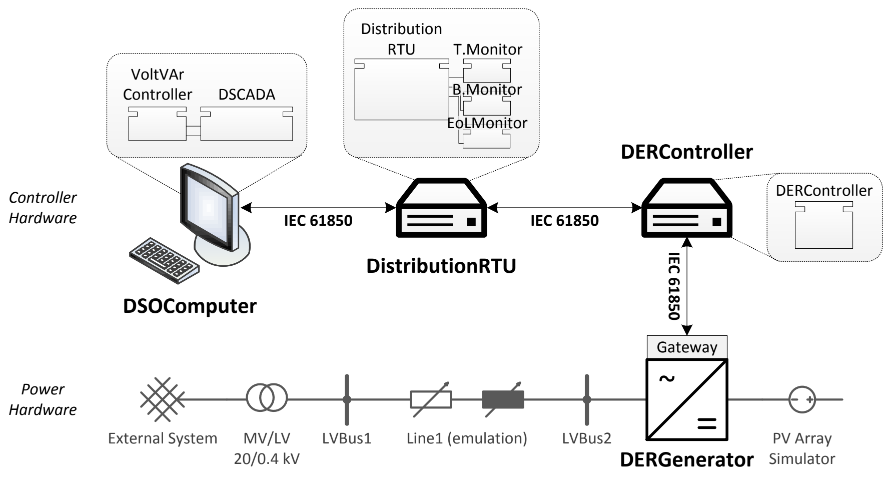


Figure 6.9: Laboratory setup for the use case validation.

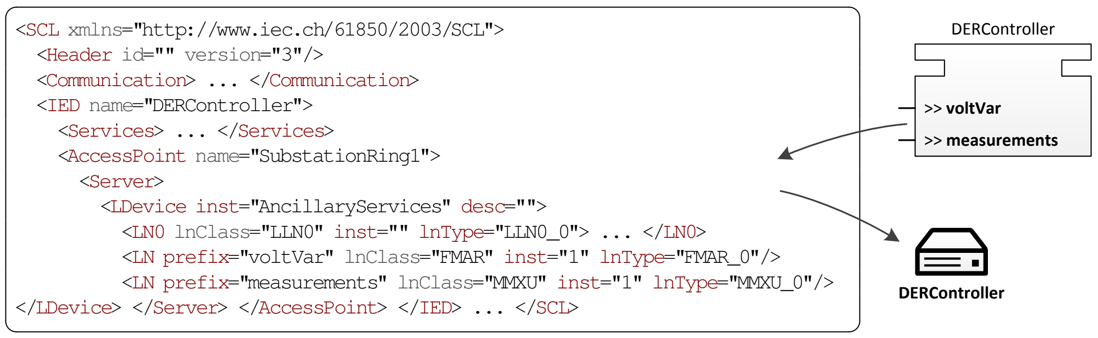
The *DERGenerator* is a commercial off-the-shelf PV inverter connected on the DC side to a PV array simulator and on the AC side to the normal LV grid. To create a dependency between the voltage and the output of the inverter, a line impedance was emulated by the laboratory equipment. Natively, the inverter has a Modbus TCP/IP control and measurement interface, but it does not have an IEC 61850 interface. To remedy this, a previously developed simple IEC 61850/Modbus gateway was used in the *DERGenerator* [152]. The gateway translates all messages between Modbus and IEC 61850. For example, measurements from the inverter (e.g., voltage, power, reactive power) are translated into IEC 61850 measurements before they are sent to the *DERController*.

Performed Experiment and Results

Before the laboratory validation can be started the application must be deployed. The first precondition for this is that the inverter is connected and feeding power into the grid. Secondly, the Devices^{499} must be started and 4diac-rte must be executed on all Devices^{499} and awaiting a deployment from the 4diac-ide. The next step is to deploy the communication configurations.

Once the SubApps^{499} from Figure 6.9 have been mapped, communication infrastructure can be generated for connections between SubApps^{499} of different Resources^{499}. Generated are both communication FBs^{499} as well as communication configurations, as described in Section 4.6.1. For this test case, all the `Interfaces` are derived from IEC 61850, which results in generation of SCL files for the communication configuration. The resulting IEC 61850 configuration for the *DERController* is seen in Figure 6.10. The *IED*^{850} is named *DERController* after the Device^{499}, and the *LN*s^{850} are prefixed with the name of the Socket^{499} interfaces.

After the SCL files have been downloaded to the Devices^{499}, the SubApps^{499} are deployed



```
<SCL xmlns="http://www.iec.ch/61850/2003/SCL">
  <Header id="" version="3"/>
  <Communication> ... </Communication>
  <IED name="DERController">
    <Services> ... </Services>
    <AccessPoint name="SubstationRing1">
      <Server>
        <LDevice inst="AncillaryServices" desc="">
          <LN0 lnClass="LLN0" inst="" lnType="LLN0_0"> ... </LN0>
          <LN prefix="voltVar" lnClass="FMAR" inst="1" lnType="FMAR_0"/>
          <LN prefix="measurements" lnClass="MMXU" inst="1" lnType="MMXU_0"/>
</LDevice> </Server> </AccessPoint> </IED> ... </SCL>
```

Figure 6.10: Generated SCL file for configuration of the IEC 61850 server in *DERController*.

according to Figure 6.9. This deployment is done using standard IEC 61499 methods and is a built-in feature of 4diac. Therefore, it is also not necessary to generate any new code from the IEC 61499 model. After a deployment is made, the Application[499] is automatically started, whereupon the SCL files are loaded and the IEC 61850 communication is initialized. For example, the IEC 61850 client in *DistributionRTU* connects to the IEC 61850 server in the *DERController*.

Once the deployment is finished, the *VoltVArController* algorithm can be validated. For this validation, the inverter is configured to an active power output of 18 kW and a reactive power output of 0 kVAr. The voltage measured by the inverter (i.e., the *DERGenerator* is forwarded to the *VoltVArController* at the *vDER* data output of the *gridStatus* FB[499] (see Figure 6.5). The other voltage measurements (i.e., *vVRC*, *vCBC*, and *cEOLM*) are all emulated and fixed to 0.9 per unit (p.u.). After stabilization, this results in a *vDER* voltage of around 1.004 p.u., and no extra reactive power. This is seen in the beginning of the time series in the top graph of Figure 6.11, where Q is the reactive power of the inverter and U is the measured voltage by the inverter (i.e., *vDER*).

In order to trigger the *VoltVArController* algorithm, a voltage spread increase is emulated. This happens at the first event *(i)*, as depicted in Figure 6.11 after 158 s. At this event, the fixed voltage of *vVRC* was changed from 0.9 p.u. to 0.86 p.u. This increases the voltage spread ($U_{max} - U_{min}$ in the bottom graph of Figure 6.11) above the allowed threshold of 0.14. This is detected by the algorithm in the *VoltVArController* and new volt-VAr curve parameters are calculated by the *ChangeDeadband* FB[499] in Figure 6.5. The new curve parameters are sent to the *DERController*, which results in a new reactive power set point for the *DERGenerator*. With the new volt-VAr parameters, the inverter starts producing reactive power. This is shown as event *(ii)* in Figure 6.11. However, since the voltage spread is still too high, the volt-VAr parameters are changed by the *VoltVArController* once again after 185 s, event *(iii)* in Figure 6.11.

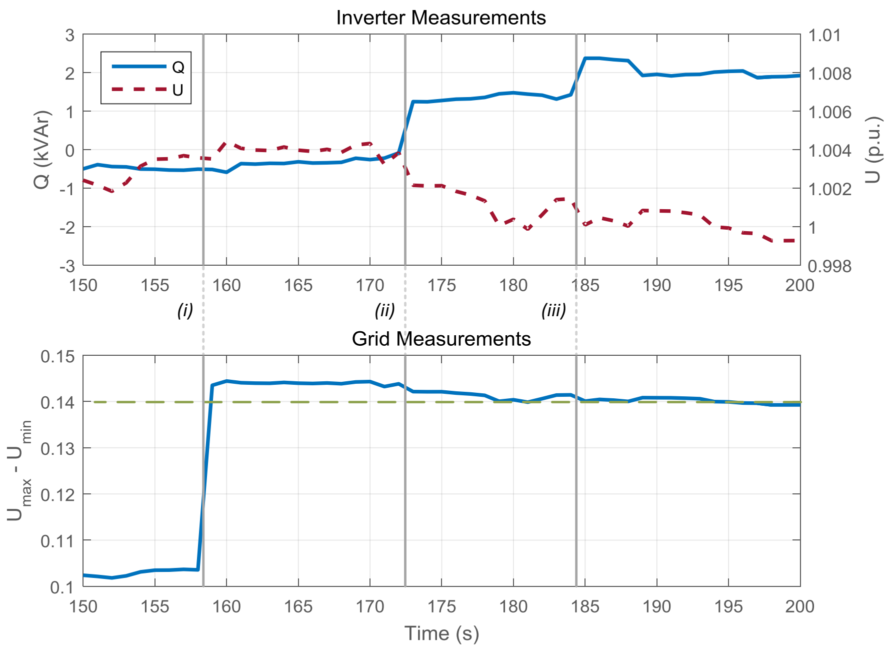


Figure 6.11: Measurements from the laboratory validation: *(i)* Detection of voltage band violation; *(ii)* First curve correction; *(iii)* Second curve correction.

CHAPTER 7

Evaluation and Conclusions

The previous three chapters have provided a theoretical description of the rapid engineering methodology in Chapter 4, a possible prototypical implementation in Chapter 5, and an application of the developed method on a test case in the previous Chapter 6. Although some of the advantages of a rapid engineering method are obvious by only studying the usage of the approach, a scientific evaluation is still needed. This evaluation represents the last step of the scientific research method presented in Section 1.3. As described there, in order to properly answer the research question of this thesis it must be shown that the developed method reduces the amount of manual effort compared to traditional smart grid engineering methods. This chapter contains this evaluation. Furthermore, the developed rapid engineering methodology is also evaluated against the specified requirements in Section 3.2.2.

This chapter also contains the conclusions of this thesis. They contain a summary of the main achievements in this work together with a description how this fits with the scientific research method chosen in Section 1.3. Finally, an outlook is made into future developments and applications of the rapid engineering methodology.

7.1 Evaluating the Rapid Engineering Methodology

The evaluation of the rapid engineering methodology is twofold. First it is made sure that the methodology fulfills all the requirements identified in Table 3.2. Secondly, and most important, is to validate the rapid engineering methodology against the main research question of this thesis. In fact, the Research Hypothesis already states a claim for how the research question can be answered. Accordingly, the validation of the rapid engineering methodology focuses on proving that hypothesis.

7.1.1 Fulfillment of Requirements

In Table 3.2, a number of requirements were presented. These were identified, based on the business cases and the thesis' research question, as the main requirements for the rapid engineering methodology. During the design of the methodology in Chapter 4, these were consistently used in order to support the design decisions. In this section, these requirements are revisited and it is shown how they are fulfilled by the application of the rapid engineering method in Chapter 6.

The first four requirements in Table 3.2 are concerned with the different phases of the engineering process. R1-Design and Specification specifies the requirement to allow design and specification in a formal way, compatible with SGAM and IEC 62559. This is shown in Section 6.2, where the test case design is explained in detail. The second requirement R2-Implementation should allow the user to implement functionality based on the design. This was also shown in detail in Section 6.3, both for the DER and for the DSO control functionalities. The third and fourth requirements, R3-Testing and Validation and R4-Release and Deployment concern the *validation and deployment* phase. According to them it should be possible to validate the developed application both through simulative and laboratory tests. It should also be possible to create a PSM and deployment this to a hardware platform. The fulfillment of both these requirements was shown in Section 6.4, where the implemented application was successfully validated and deployed.

The following two requirements are focusing on the performance of the rapid engineering methodology. Requirement R5-Seamlessness should assure that the transition between one engineering step to another is as seamless as possible. The purpose of R6-Rapidness and Effort is to improve the rapidness of traditional engineering methods and thus reduce the development efforts. The fulfillment of these requirements is directly related to the automation using MDE techniques. These automation possibilities are clearly shown during the application in Chapter 6. Furthermore, these requirements are also related to the Research Hypothesis of this thesis, which is discussed more below in Section 7.1.2.

Also related to the performance of the rapid engineering methodology is R7-Correctness. Its intention is to assure that the implemented application is as correct as possible. It has been shown that the automatic generation possibilities of the rapid engineering methodology saves manual work. Consequently, since less human interaction is needed this will also reduce the risk of human errors for these tasks. Furthermore, the application of the methodology to the test case in Chapter 6 with the included validation also shows that the intended functionality was correctly implemented.

The purpose of R8-Domain Expertise is to allow experts from different domains to participate in the engineering process. This has been shown with the integration of different DMs, as explained in Section 4.2. Compatibility with SGAM and IEC 62559 for the use case design allows the interaction from people without any specific technical knowledge. In Section 4.3.2, it was explained how the OMG IDL was integrated with PSAL. This is a well known modeling language for information exchange. In order to represent power system models, CIM was used, as explained in Section 4.3.2. Finally, for automation purposes IEC 61499 was used (see Section 4.4).

The requirement R9-Handling Legacy Systems specifies that it should be possible to integrate legacy systems in both directions—applications developed with the rapid engineering method should be integrable with an existing system, and the configuration and functionality of legacy components should be integrable into the rapid engineering method. Section 6.4.2 shows that existing third-party `Functions` can be automatically handled by the engineering method. On the other hand, integration of existing configurations was shown in Section 4.4.3, where it was described how SCL files can be imported.

Requirement R10-Interoperability is similar to the previous two requirements since it is also concerned with interaction. Interoperability should be assured on all levels, from specifications over implementation, to deployment and finally during operation. For the rapid engineering methodology, this has first of all been shown with compatibility with SGAM and IEC 62559 shown in Section 4.3.1. Secondly, the possibility to import and export domain specifications (e.g., IEC 61850, CIM) also shows that interoperability with existing standards is given. This is shown in Section 6.4.1 and Section 6.4.2.

Another requirement that regards the interaction with the rapid engineering method is R11-Changing Requirements. According to this requirement, it should be possible to consider new requirements at any time during the engineering process. This has been specifically shown in Section 6.4.1, where the communication protocols were changed without the need for any other changes.

In summary, it has been shown that all specified requirements have been fulfilled by the rapid engineering methodology. Furthermore, it is also provides solutions to the presented business cases in Section 3.1. These cases were used to explicitly show the potential of a rapid engineering process for a selected number of stakeholders. Therefore, the developed rapid engineering methodology is not only a solution to the problems presented in the business cases, but also an opportunity for these stakeholders to improve their engineering processes.

7.1.2 Research Validation

The second part of the evaluation is to validate the rapid engineering methodology against the research question: *Given the traditional engineering process for smart grid applications—covering use case design, implementation, validation, and deployment—what can be done to significantly reduce the amount of manual work needed from the smart grid engineer(s)?* In Section 1.2, a research hypothesis was formulated that already hints at a solution for this question. Thus, the validation in this section will focus on proving that this hypothesis holds.

According to the Research Hypothesis *MDE of smart grid applications, with model transformations during the engineering phases—and for the transitions between the different phases—will reduce the amount of manual work needed to describe information* *in multiple models.* If this hypothesis holds it is also one possible answer to the research question. However, in order to show that the hypothesis holds the rapid engineering methodology must be compared to traditional solutions. At the moment, no smart grid engineering approaches exist that can be directly compared to the method in this thesis. But, nevertheless existing solutions for each single development phase can be used together to compare with the rapid engineering method. For this validation, two different solutions are chosen.

Comparison to IEC 62559 and Automation Programming Languages

The first traditional engineering approach uses the IEC 62559 templates for use case design and IEC 61131 for the *implementation* phase. As with SGAM, IEC 62559 is one of the most frequently used templates for use case descriptions of smart grid applications. Typical approaches for industrial automation are IEC 61131 and the already well-known IEC 61499. But, since IEC 61131 is more common in today's systems than IEC 61499 it will be used for the comparison with the rapid engineering methodology.

In this approach, the IEC 62559 use case template is used for the *design* phase. The test case from this thesis is available in Appendix B.4 and it shows how use cases are described using the template. The main difference compared to PSAL is that, although IEC 62599 uses tables to structure the description, it is not a formal and machine-readable approach. Instead mainly prose is used to describe the use case (which is also possible to do with comments in PSAL). Due to this, using the use case information directly for an automated approach would only be possible if the IEC 62559 template was accompanied with further design rules. For example, it would be theoretically possible to extract IEC 61850 LNs if the exact names of the LNs were used by the engineer for the information exchanged (compare Table B.54). For this comparison, it is assumed that no extra design rules are used. Therefore, in this case the only real advantage with IEC 62559 is its simplicity since no special tools are necessary.

Since it is not possible to automate the transition from from the IEC 62559 template, much of the work already done during the *design* phase needs to be repeated during the *implementation* phase. Thus, each functionality defined in the use case description needs to be implemented using IEC 61131. Furthermore, also any communication information described in the use case (i.e., information exchanged) must be transfered into an appropriate representation in IEC 61131. In summary, the transition between *design* and *implementation* requires significantly more manual work with the traditional engineering approach than with the rapid engineering method.

The IEC 61131-3 part also provides FBs. In fact, the FBs in IEC 61499 are an extension of FBs in IEC 61131-3, as already stated in Section 2.3.2. Let us assume that the engineer has created empty IEC 611313 FBs for each functionality described in the use case. Then the next step is to implement the functionality of each FB and to connect them with each other. There already exist a number of comparisons in the literature, where the usability of IEC 61131 and IEC 61499 for smart grids is studied [171, 147]. As already discussed in Section 4.4.1, one of the advantages of IEC 61499 is its application centered modeling approach [177]. It is also independent of where the FBs are executed. With IEC 61131, there is no such approach. Instead, a device centered approach is used, where a programmed functionality is always platform-specific. Therefore, there is also no application view available in IEC 61131 (compare with Figure 6.2). More detailed studies of the differences between IEC 61131 and IEC 61499 have already been done [168, 169, 171, 147, 177], and would go beyond the scope of this thesis. Nevertheless, despite their differences the amount of manual work needed for the implementation phase is similar for both IEC 61131 and the rapid engineering method.

The rapid engineering methodology also offers the possibility to automatically generate communication configurations based on the PSAL design. However, for the traditional approach, the same conditions apply as before. Since the IEC 62559 template is no machine-readable format, the available communication information in the use case description cannot be directly used. Therefore, it is also not possible to generate communication configurations (e.g., SCL files). Consequently, these configurations need to be manually created based on the information in the use case description. The result is significantly more manual effort and an increased risk for human errors. Furthermore, it also not possible to generate simulation models from the IEC 62559 template. Thus, if a simulative validation is needed these also need to be manually created.

For the validation and deployment phase, the manual effort is comparable between the two approaches. In both cases, the actual validation is done manually. Furthermore, although the IEC 61131 standard does not specify any deployment process many compatible software tools do. Therefore, both the traditional and the rapid engineering approach require about the same amount of manual work.

Summarizing the comparison between the traditional approach, using IEC 62559 together with IEC 61131 and the rapid engineering method, a number of differences in manual effort were found. Using the traditional engineering approach, the use case would is only used as a specification. Based on this specification, a new implementation in IEC 61131 would be created. Consequently, much of the work is done at least two times. A further disadvantage is that any changes in the use case description require manual changes in the implementation. Finally, another solution based on IEC 61131 would also not support the automatic generation of communication configurations. Listed, the main differences are:

- Use case descriptions with IEC 62559 cannot be easily used in an automated approach and can thus merely be used as a specification.

- Functional and communication design need to be repeated multiple times with the traditional approach, whereas with the rapid engineering method this is only done once during the *design* phase.

- Succeeding changes in the IEC 62559 use case description require manual changes in the implemented, while changes in the PSAL description are automatically transfered into the implementation.

Comparison to SGAM and General-Purpose Programming Languages

The second traditional engineering approach uses the SGAM Toolbox [31] for the *design* phase and general-purpose programming language (e.g., Java, C++) for the *implementation* phase. As already discussed in Section 2.2.2, the SGAM Toolbox is a commonly used tool for use case design according to the SGAM approach. It is also widely accepted among the stakeholders in Table 3.1. It is also very common that general-purpose programming languages are used for the implementation. Furthermore, it is also assumed that the programming language is accompanied with some sort of Continuous Integration (CI) process. This means that developed functions can be integrated and deployed to the field devices in an automated manner.

The SGAM Toolbox follows a graphical approach for the design phase. Each layer is modeled using SGAM specific, or normal UML artifacts [31]. Compared to the textual approach of PSAL, a graphical approach is better to understand spatial layouts. For example, it is easier to understand how components are connected with each other through a graphical view of the component layer of SGAM, see Figure 6.1, than a textual description, see Listing 6.7. Nevertheless, once the user is accustomed to a textual notation, the design is usually faster [110]. Both approaches have their own advantages and disadvantages but in short, the amount of work needed by the user is similar. Furthermore, with both methods it is possible to completely model the test case in Appendix B.4. Thus, it is assumed that after the *design* phase the same information is available in both engineering methods.

Before the *implementation* phase, a closer study of the transition from the *design* phase should be made. With the rapid engineering methodology SubApps^{499} are automatically generated based on the `Functions` in the PSAL description. The SGAM Toolbox is part of the Enterprise Architect tool, which also offers code generation support. Based on modeled UML diagrams code for a number of general-purpose programming languages can be generated. But, there is still a number of differences between the two approaches.

In order for Enterprise Architect to successfully generate code, the user has to create the UML models in a certain way. Therefore, even if a detailed UML model is available it may not be possible to generate code from it because it was not modeled in the right way. With PSAL, this is not the case. The syntax of PSAL is more constrained and it is optimized to allow function generation. Thus, it is always possible to generate IEC 61499 code from a PSAL `Application`.

The rapid engineering methodology also allows for a more detailed transfer of communication information from the *design* to the *implementation* phase. The `Interfaces` and `Events` are incorporated into the SubApps^{499} and it is even possible to model different communication patterns (i.e., publish/subscribe and client/server). This is not possible with Enterprise Architect. It is possible to model information exchange through sequence diagrams, but how the information is exchanged (i.e., a local function call or a communication interface) is not considered. Therefore, with Enterprise Architect the user has to include this information manually into the generated code.

In summary, function generation is possible both in Enterprise Architect and with the rapid engineering method but more user effort is required in Enterprise Architect. A higher design effort is required since the user has to model in the right way. Also, communication patterns are not included, which requires the user to include these manually after the code has been generated.

During the *implementation* phase, both approaches require similar amount of manual work. In both cases, the functionality must be manually implemented. The automatic implementation of functions described in Section 4.5.1 is not considered for the comparison. It is assumed that the implemented functions are new and have no previous match.

For the transition between the *implementation* and the *validation and deployment* phase, the rapid engineering methodology provides a number of automation services. Communication interfaces are automatically configured with automatically generated configurations (e.g., SCL files). Moreover, the implemented functions are automatically connected to the communication interfaces, see Section 4.6.1. Also, from the PSAL description, simulation models can be automatically generated. None of these automation possibilities exist with Enterprise Architect. This means significantly more manual work for the engineer. Communication interfaces must be configured and integrating with the functions. This is time consuming and also requires detailed knowledge about the used communication protocols. If a simulative validation is made the engineer also has to create a simulation model.

The last phase is the *validation and deployment*. For both engineering approaches, the validation is made manually. For example, for a simulative validation the simulation models—automatically generated or manually created—are simulated with a power system analysis tool (e.g., PowerFactory). With the assumption that a CI process is used for the traditional engineering approach, also the deployment is similar for both approaches. Consequently, there is no big difference in the amount of manual work for this phase using one or the other of the engineering approaches.

In summary, compared to a traditional smart grid engineering approach, using the SGAM Toolbox and general-purpose programming languages, the amount of manual work can be significantly reduced using the rapid engineering methodology. Although the SGAM Toolbox and especially Enterprise Architect support code generation, this code is generic and not specialized for smart grid applications. Furthermore, the generation of communication configurations is not supported by the SGAM Toolbox. It is also not possible to use generic communication patterns as provided by IEC 61499. Therefore, they again have to be implemented manually and integrated into the source code. The main differences are summarized as:

- PSAL was especially created to allow code generation, which guides the user to create designs suitable for code generation.
- With the rapid engineering methodology, communication and interface information only have to be described once during the design, compared to the traditional approach where this has to be done repeatedly during all the phases.
- System and component information designed with PSAL can be directly used to generate simulation models, whereas with the traditional engineering approach these have to be manually created.

Validation Summary

The main research question of this thesis is to answer how it is possible to reduce the amount of manual work with an automated rapid engineering method, which covers the whole development chain for smart grid automation applications—from use case design to its field deployment. To tackle this, a Research Hypothesis was formulated, which states that using MDE technologies for the smart grid engineering method is one answer to the research question. The rapid engineering methodology developed in this thesis is one proof that this hypothesis holds. To further highlight this fact, the previous two sections have shown a comparison with other traditional engineering methods.

The comparison with the other engineering methods has especially shown one thing. With traditional engineering methods, a significant manual effort is needed to repeatedly copy and recreate artifacts that were already specified during the *design* phase. One example is the functionality of the application and how information is exchanged between functions. Another example is the configuration of communication interfaces. With the rapid engineering methodology, this manual effort is reduced to a minimum since these artifacts are automatically reused in the following engineering phases. In total, the complete development process is summarized to the following steps:

1. Initial design of the `Application` and the `System` using PSAL
2. Detailed design of `Interfaces` and `Events` using protocol mappings
3. *Transformation into IEC 61499 Application^{499} and System^{499} models*
4. Function design by implementation of SubAppTypes^{499}
5. *Generating and downloading the communication configurations (e.g., SCL files)*
6. *Downloading the Resources^{499} to their Devices^{499}*
7. Validation and/or field execution

From these steps, only the first, second, fourth, and the last step requires significant manual inputs from the engineer. The other steps are automated and only require a simple activation by the user. The application of the rapid engineering method in Chapter 6 also shows that these steps are enough to implement a complete use case. The automation of these steps is possible through the use of MDE methods, in this case specifically model-to-model transformations. They allow a transformation and reuse of previously defined information and thus provide the automatism of the rapid engineering method. Without the automatic transformations, these steps would require manual effort. Consequently, this also proves the soundness of the Research Hypothesis and, as a result, also provides an answer to the main research question in Section 1.2.

7.1.3 Comparison with State of the Art

One of the main goals of a PhD is that it should make a significant contribution to the current state of the art. Hopefully, this contribution is also an improvement compared to existing solutions. In Chapter 2 both background knowledge and possible methods and solutions are presented. Existing engineering approaches from the smart grid domain are complemented with an outlook towards other domains and the tools used there. At the end of Section 2.6 a number of missing features and gaps were identified. These features are what is missing in order to solve the main research question of this thesis—to improve the traditional engineering methods in terms of manual effort. In Table 7.1 the results of this work is compared with the missing features and current gaps identified in Chapter 2.

Table 7.1: Comparison with state of the art and existing approaches.

Gap	Description	Comparison with this work
Holistic engineering	Non of the existing approaches cover the whole development process (design, implementation, validation, and deployment) in an integrated manner.	Compared to other existing solutions this missing feature is covered by the rapid engineering methodology. All engineering phases are considered in one holistic approach.
Model-based	Model-based engineering concepts for smart grids are missing or only partly available.	The rapid engineering method is based on an MDE concept covering all phases. This allows for automation with MDE techniques like model transformations.
Effortless transitions	At the moment there is no method that completely removes the effort of moving from one engineering step to the next.	Model transformation also achieves automatic transitions between the engineering phases. This way information from one phase can be automatically reused in the next.

System of systems	Current engineering methods are primarily focusing on the development of single systems and not a system of systems (e.g., one focus on the development of substation automation).	The application centered design approach of the rapid engineering methodology supports development of applications in an holistic manner. Focus is put on the whole solution, not only the implementation for each separate system.
Multiple domains	No approach is currently available that integrates general domain knowledge from different domains, such as control, communication, or power system modeling.	Especially the relationship of PSAL with well known domain models closes this gap. This feature also allows import and export of already existing models (e.g., communication models).
Integration with legacy systems	Many of the current approaches do not offer any support for the integration of legacy systems.	Due to possibility of integrating new models into PSAL this feature can be easily covered. By defining new model transformations it is even possible to integrate proprietary models.
Smart grid domain specific	Approaches that are optimized for smart grid engineering are only partly available.	The rapid engineering methodology is a domain specific approach intended for smart grids. Not only during the design phase, but throughout the whole engineering process the connection to smart grids is clear.

7.2 Conclusions

This section concludes the thesis. A summary of the previous chapters is given and finally, the outcome of the work is critically analyzed with an outlook towards future work.

7.2.1 Recapitulation

With the smart grid rollout new intelligent concepts for measurement, control, and automation are being implemented in the power systems today. Nevertheless, the smart grid system is an evolving system. Focus is moving from a single system to a system of systems perspective. As a result, the engineering complexity is increasing, which in the end also means increasing costs. To tackle this, there is a need for smart and automated engineering methods, also in the smart grid domain.

This work fulfills this need with a rapid engineering methodology especially designed for the smart grid domain. The goal of the methodology is to automate the engineering process, thus helping and supporting engineers. In detail, the automation was provided using MDE techniques, otherwise mostly seen in software engineering. MDE focuses on the development of models and the automatic transformation between these models. Thus, if the transition through an engineering process is seen as a transformation between different engineering models, the automation of these transformations will also lead to an improved engineering process.

This thesis follows an inductive-hypothetical research strategy. It consists of five steps, starting with the *initiation*, where the state of the art is studied in order to form a descriptive empirical model. In this work the current state of the art for smart grid engineering has been presented in Chapter 2. It shows an overview of current methods used in the smart grids domain, but also gives an outlook to what approaches are used in other domains.

In order to formalize a rapid engineering methodology it must be clear for whom this process is intended and also what their requirements are. Therefore, the next step of the work was a use case study and requirement analysis. This part represents the second step of the research strategy: the *abstraction*. The goal is to substantiate the issues identified during the *initiation* through field studies. In this case this is represented by three business cases, collected from different research projects. Each business case is associated to one or more stakeholders. Furthermore, for each business case an example use case was shown. The business cases, stakeholders, and use cases served as a basis for the identification of the requirements for the rapid engineering methodology. These requirements make up the descriptive conceptual model which is the result of the *abstraction* phase.

Based on the requirements, the next step is the *theory formulation*, the third step of the research methodology. In the case of this work, the goal of this step is to formulate a theory for the best way to improve the traditional engineering methods for smart grids. The result was the rapid engineering methodology, which utilizes MDE in order to automate the smart grid engineering process. In the end, four engineering phases were crystallized:

1. *Design* of the use case and specification of the application that is to be implemented
2. *Implementation* of the application including generation of executable code as well as communication configurations
3. *Validation* of the developed application, either using simulations or laboratory tests
4. *Deployment* of the generated code and configurations to field devices

In order to follow an MDE approach, PSAL was created for the *design* phase. It is a DSL that supports use case design according to the SGAM method. Specifically, the design of a smart grid use case is organized into five layers: business, function, information, communication, and component layer. For this purpose, PSAL provides different constructs. In summary, the main goal is to provide a formal method for SGAM compatible and platform independent use case descriptions of smart grid applications.

The second step was to combine PSAL with IEC 61499 in order to support detailed function development in the *implementation* phase. Any functionality initially designed and specified using PSAL in the first phase is implemented using IEC 61499. As engineering support, an automatic transformation from the function layer of the PSAL specification into IEC 61499 Application^{499} and System^{499} models is provided. Furthermore, automatic generation of functionality is also offered. Previous user implementations are compared to the specifications of new functions. If a match is found the previously implemented function can be used as a template by the user.

Based on the IEC 61499 implementations and the PSAL descriptions, three types of code are generated: platform specific code, communication configurations, and simulation models. The platform specific code is an adaptation of the implemented functionality based on the execution platform. Secondly, different communication configurations are generated, such as IEC 61850 SCL files. These are based on both the information and communication layers of the PSAL specification. Finally, simulation models can also be generated. Based on the component layer in the PSAL description a CIM model is generated, which can be imported and used in a power system simulator.

Figure 7.1 shows an overview of the main model transformations that are possible with the rapid engineering methodology. In general, if a machine readable format for SGAM or IEC 62559 was available such descriptions could be imported and used as a starting point for the process. Conversely a PSAL description could also be used to create descriptions in SGAM or IEC 62559. Furthermore, CIM and SCL configurations can also be used as inputs for PSAL. This allows import of existing power system models and communication configurations. In the end, executable IEC 61499 code is generated together with any needed CIM models and communication configurations (e.g., SCL files).

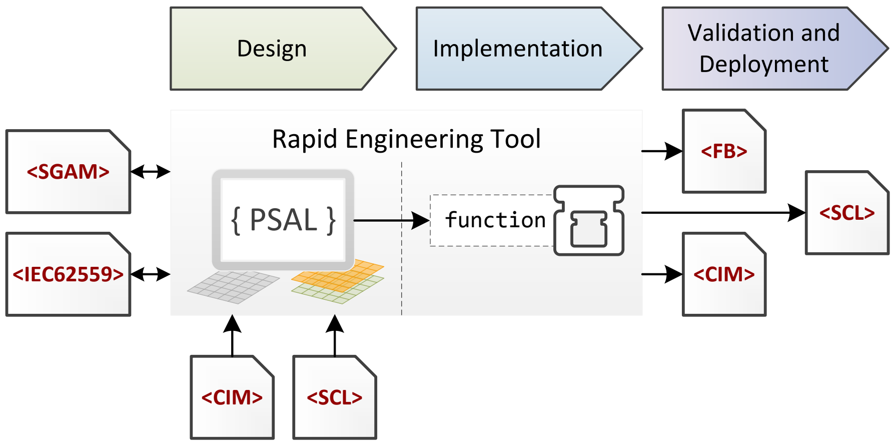


Figure 7.1: Overview of the main model transformations possible with the rapid engineering methodology.

Validation is either made using a simulative or a laboratory approach. It is followed by the *deployment* phase where the generated code is downloaded and executed on compatible field devices. For this purpose the IEC 61499 deployment approach is used. It provides a download process, where the implemented Application[499] is downloaded to field devices.

Once the theory for the rapid engineering methodology is formulated, the fourth step of the research strategy follows—the *implementation*. The main goal of this step is to show how the formulated theory can be implemented and used on the studied cases during the *abstraction* phase. First of all, a prototypical implementation of the rapid engineering methodology was developed. Using an iterative process, the results was an Eclipse based toolkit. Using only this tool, it is possible for the engineer to go through all the engineering phases described above. For the second part, the rapid engineering methodology was applied to a specific test case, similar to the studied business cases. The test case concerns the implementation of a coordinated volt-VAr control and includes examples and detailed explanation of all the engineering phases. Furthermore, to show different validation scenarios both a simulative and a laboratory test are used to validate the implemented functionality.

The fifth and last part of the research strategy is the *validation*. In this part results from the *implementation* part are compared to the existing state of the art. For this work the evaluation of the rapid engineering methodology is threefold. First, the methodology was compared to the requirements. Secondly, and most important, the rapid engineering methodology was validated against the main research question and the hypothesis of the thesis. Specifically, the goal was to show how applying MDE technologies to the traditional smart grid engineering process can reduce the manual development effort. Thirdly, the developed methodology was compared to the existing state of the art.

To confirm the hypothesis of the thesis, the rapid engineering methodology was compared to two other solutions. The first solution is a traditional engineering methodology, where the use case is documented using the IEC 62559 template. With this as a basis, a new implementation is created. Consequently, much of the work is done at least two times. Furthermore, changes in the use case description require manual updates in the implementation. A comparison to a more advanced method, the SGAM Toolbox [31], also shows some important differences. Although code generation is supported by the SGAM Toolbox, this code is generic and not specialized for smart grid applications. Another important advantage of the rapid engineering methodology is the generation of communication configurations. These configurations are currently not supported by the SGAM Toolbox, which also does not provide a rapid deployment process.

The comparison with the other methods has especially shown one thing. Traditional engineering methods need significant more manual effort. This is needed to repeatedly copy and recreate artifacts that were specified during the *design* phase. With the rapid engineering methodology, this manual effort is reduced to a minimum since these artifacts are automatically reused in the following engineering phases. Summarizing, the development process is represented with the following steps:

1. Design and specification using PSAL
2. *Transformation into IEC 61499 models*
3. Function implementation using IEC 61499
4. *Generating and downloading the communication configurations*
5. *Downloading the implemented functionality*
6. Validation and/or field execution

From these steps, only the first, third, and the last step requires manual inputs from the engineer. The other steps are automated and only require a simple user activation. MDE methods, specifically model-to-model transformations, provide a powerful approach to automate these steps. Information can be automatically reused throughout the rapid engineering methodology, which otherwise would require manual effort. Consequently, this also proves that the main goal of the thesis has been achieved.

7.2.2 Reflection and Future Work

As always when this kind of study is made, only a portion of all possible cases can be studied. Consequently, the rapid engineering methodology as presented here should be seen as a first attempt and is still far from all-encompassing. This section selectively points out a number of issues, where future work should be invested.

First of all, during the design of PSAL only a subset of future modeling needs was covered. Much focus was appointed to model information exchange in a detailed manner. This is also clearly seen throughout the thesis. Other issues are not covered in the same detail, or not at all—at least not explicitly. Topics of interest could for example be timing, system performance, or functional correctness. Moreover, future work could include explicit modeling of these issues as well as automatic checks. In summary, the current state is not to be considered final, but instead as a starting point for further studies.

Another open issue of the current methodology, and especially the prototype, is that it does not support round-trip engineering in an optimal way. If completely possible it would allow the engineer to start anywhere on the engineering process and generate artifacts in both directions. For instance, a user could start directly with the *implementation* phase using IEC 61499 and based on this generate the PSAL specification. At the moment, this is not completely supported. In theory, the model transformations defined in the methodology support it. The issue here is rather the lack of means to express all information in all models (e.g., IEC 61499 is not intended for modeling of power system components, and would only allow it with great difficulty). Additionally, the current prototype was not designed to handle this case.

Scalability is another matter not directly handled by the rapid engineering methodology. As already mentioned in the introduction of Chapter 4, handling multiple use cases at once is not a goal of the thesis. Nevertheless, also with only one use case scalability may be an issue. This is currently not handled by the methodology. For example, no constructs were considered to allow import of existing specifications. Furthermore, no constructs were created to allow specification of multiple `Components` at once (e.g., multiple measurement devices of the same brand). The same can be said about the prototype. Also in this case more attention to scalability must be paid for a industry prototype. Consequently, this is another matter where future research is needed.

Currently, the *validation* phase is only partly automated. One direction for future work could be to also provide automation support for testing and validation, as indicated by Strasser et al. [153]. Of course, this would also require further solutions in the *design* phase for specification of validation scenarios and expected results.

The rapid engineering methodology is an example of active engineering support provided to the user. Of course, MDE technologies is only one possibility of how such support can be realized. A further solution could be to use machine learning or expert systems approaches to additionally improve the engineering support. By studying the user engineering behavior, the machine could learn how to best implement smart grid applications, and in a further step use this knowledge to instruct new engineers. The current automatic implementation of functions, described in Section 4.5.1, is one example of how to take advantage of the user experience. Similar solutions could also be possible for the information design (e.g., how to group `Interfaces` and `Events` into `Modules`).

Concluding, future work will have to prove the feasibility of the rapid engineering methodology for large-scale applications. Nevertheless, with this work a big step is made towards a comprehensive engineering support for the development of smart grid applications. There is still much work to be done, but the rapid engineering methodology has shown that with the right tools the amount of manual effort for developing smart grid applications can be significantly improved.

Appendices

APPENDIX A

PSAL Grammar

The grammar of PSAL using an EBNF notation [33].

```
PSAL ::= PsalContent*
PsalContent ::= System | Application | Module

/* System grammar */
System ::= 'system' ID '{' SystemContent* '}'
SystemContent ::= Component | Connection
Component ::= ICTComponent | ElectricalComponent
ICTComponent ::= Device | OtherICTComponent
Device ::= 'device' ID '{' DeviceContent* '}'
DeviceContent ::= Resource | ICTInterface
Resource ::= 'resource' ID
OtherICTComponent ::= OtherICTComponentTypes ID IDValuePairBody?
OtherICTComponentTypes ::= 'gateway' | 'switch' | 'router'
PhysicalInterface ::= ICTInterface | Terminal
ICTInterface ::= ICTInterfaceType ID IDValuePairBody?
ICTInterfaceType ::= 'ethernet' | 'wireless' | 'serial' | 'analogue' | 'digital'
Terminal ::= 'terminal' ID IDValuePairBody?
ElectricalComponent ::= Transformer | ElectricalEquipment
Transformer ::= 'transformer' ID '{' TransformerWinding+ '}'
TransformerWinding ::= 'winding' ID '{' ElectricalEquipmentContent* '}'
ElectricalEquipment ::= ElectricalEquipmentType ID '{' ElectricalEquipmentContent* '}'
ElectricalEquipmentType ::= 'generator' | 'line' | 'eswitch' | 'consumer' | 'busbar'
ElectricalEquipmentContent ::= Resource | PhysicalInterface | IDValuePair
Connection ::= 'connect' QualifiedName 'with' QualifiedName IDValuePairBody?

/* Application grammar */
Application ::= 'application' ID '{' ApplicationContent* '}'
ApplicationContent ::= Function | Connection | Module
Function ::= 'function' ID (FunctionMapping)? '{' FunctionContent* '}'
FunctionMapping ::= 'at' QualifiedName
FunctionContent ::= Function | Connection | ServiceImplementation
ServiceImplementation ::= ProvidedService | RequestedService
```

```
ProvidedService ::= ProvidedIDLInterface | ProvidedIDLEvent
RequestedService ::= RequestedIDLInterface | RequestedIDLEvent
ProvidedIDLInterface ::= 'provides' QualifiedName ID ('{' ParameterAssignment* '}')? ProtocolUsage?
ParameterAssignment ::= QualifiedName '=' ValueLiteral
RequestedIDLInterface ::= 'requests' QualifiedName ID ('{' ParameterAssignment* '}')?
ProvidedIDLEvent ::= 'emits' QualifiedName ID ('{' ParameterAssignment* '}')? ProtocolUsage?
RequestedIDLEvent ::= 'consumes' QualifiedName ID ('{' ParameterAssignment* '}')?
ProtocolUsage ::= 'using' QualifiedName

/* Information grammar based on [74] */
Module ::= 'module' ID '{' ModuleContent+ '}'
ModuleContent ::= Module | Interface | Event | Constant | TypeDecl
Interface ::= ('abstract')? 'interface' ID InterfaceInheritance? '{' InterfaceContent* '}'
InterfaceInheritance ::= ':' QualifiedName
InterfaceContent ::= Attribute | Operation | Constant | TypeDecl | CommunicationParameter
Attribute ::= ('readonly')? 'attribute' TypeSpecification ID (',' ID)*
Operation ::= ('oneway')? OpTypeSpecification ID OpParameters
OpTypeSpecification ::= TypeSpecification | 'void'
OpParameters ::= '(' Parameter (',' Parameter)* ')' | '(' ')'
Parameter ::= ParamAttribute TypeSpecification ID
ParamAttribute ::= 'in' | 'out' | 'inout'
Event ::= ('abstract')? 'eventtype' ID EventInheritance? '{' EventContent* '}'
EventInheritance ::= ':' QualifiedName
EventContent ::= TypeDecl | StateMember | CommunicationParameter
StateMember ::= ('public' | 'private') TypeSpecification ID
Constant ::= 'const' TypeSpecification ID '=' ValueLiteral
TypeDecl ::= EnumType | Sequence
EnumType ::= 'enum' ID '{' EnumValue (',' EnumValue)* '}'
EnumValue ::= ID
Sequence ::= 'typedef' 'sequence' '<' TypeSpecification (',' Int)? '>' ID
TypeSpecification ::= Types | QualifiedName
CommunicationParameter ::= 'parameter' ID ('=' ValueLiteral)?

/* Parameter definitions */
IDValuePairBody ::= '{' IDValuePair* '}'
IDValuePair ::= ID '=' ValueLiteral

/* Terminals and literals */
ID ::= ([a-zA-Z] | '_') ([a-zA-Z0-9] | '_')*
String ::= '"' [^"]* '"'
QualifiedName ::= ID ('.' ID)*
ValueLiteral ::= String | BooleanLiteral | NumberLiteral
BooleanLiteral ::= 'true' | 'false'
NumberLiteral ::= ('-')? (Int | FloatingPtLiteral | HexLiteral)
Int ::= [0-9]+
FloatingPtLiteral ::= INT '.' INT | INT | '.' INT
HexLiteral ::= '0' 'x' [0-9a-fA-F]+
Annotation ::= '@' ID AnnotationParams?
AnnotationParams ::= '(' AnnotationParam (',' AnnotationParam)* ')' | '(' ')'
```

```
AnnotationParam ::= ID
Types ::= 'boolean' | 'float32' | 'float64' | 'int8' | 'int16' | 'int32' | 'int64' | ↵
    ↪ 'uint8' | 'uint16' | 'uint32' | 'uint64'| 'byte' | 'word16' | 'word32' | ↵
    ↪ 'word64' | 'string'| 'date'
```

Comments can be used both as multi-line and single-line. Multi-line comments are started with `/*` and ended with `*/`. They can span multiple lines but cannot be nested. A single-line comment is started with `//` and terminates at the end of the line. Principally comments may contain any characters that do not terminate the comment.

APPENDIX B

Use Case Descriptions According to IEC 62559

B.1 Use Case Description for BC1: Use Case Design

Table B.1: Use case name [52].

Use Case Identification		
ID	**Domain(s)**	**Name of Use Case**
BC1-UC1	Distribution, DER, Consumer	Integrated Vol-VAr Control Centralized

Table B.3: Use case version [52].

Version Management				
Version No.	**Date**	**Name of author(s)**	**Changes**	**Approval Status**
A	09/16/2011	Brian D. Green	Initial release	
B	10/31/2011	Ron Cunningham	Final team review	Final version

Table B.5: Use case narrative [52].

Narrative of Use Case
Short description
The centralized integrated volt-VAr control system is an integral part of the distribution SCADA environment. One central system manages and controls all volt-VAr controller devices on the regional distribution network [52].
Complete description

This control implementation uses a centralized approach where the actual intelligence of the volt-VAr control lies within the DSO, and all actions are routed through the *D-SCADA* system. The controller devices in the field (i.e., the *Voltage Regulator Controller*, the *Capacitor Bank Controller*, the *DER Controller*, and the *End-of-Line Monitor*) send measurements, and the *Circuit Reconfiguration Controller* sends the grid status, to the *Distribution RTU*. The *Distribution RTU* aggregates this information and forwards it to the *D-SCADA* system. Also the *Powerflow Module* acquires necessary loadflow information and passes it on to the *D-SCADA*. Thereupon the *D-SCADA* forwards all vital information to the *VVC*. The *VVC* computes new set points and commands for the controller devices. These are forwarded by the *D-SCADA* and the *Distribution RTU* to the field devices. All important data are continuously sent to the *Distribution Historian* [52].

Table B.7: Use case diagram.

Diagram of Use Case

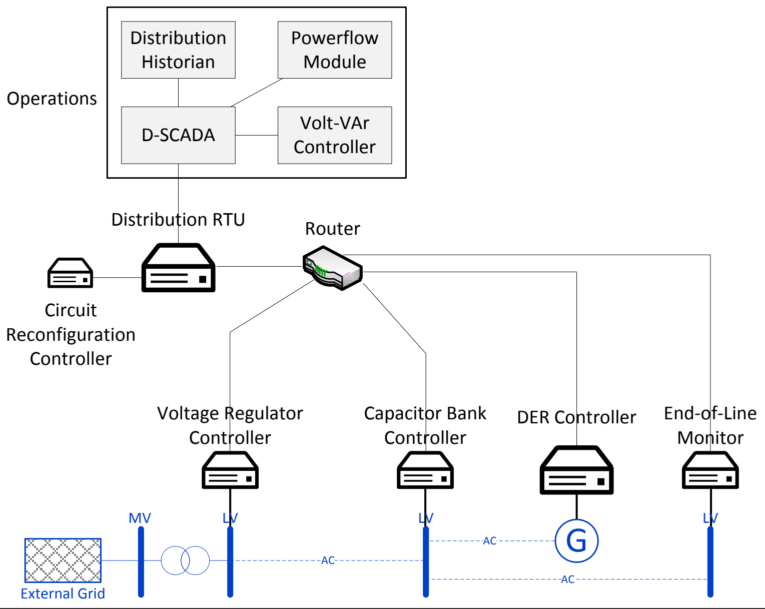


Table B.8: Use case actors [52].

Actors		
Actor Name	**Actor Type**	**Actor Description**
Capacitor Bank Controller	IED	Controller for the capacitor banks.
Circuit Reconfiguration Controller	Station Controller	Controller for the circuit reconfiguration system (logic and application)
Distribution Historian	Application	Data Historian for the distribution system
Distribution RTU	RTU	Distribution RTU
D-SCADA	SCADA	Distribution SCADA
End-of-Line Voltage Monitor	End Point Monitor	Voltage monitors for endpoints at the end of the line
Powerfow Module	Application	Loadflow calculation engine

Volt-VAr Controller	Application	Controller for volt-VAr operation
Voltage Regulator Controller	IED	Controller for the voltage regulators

Table B.10: Excerpt of the step-by-step analysis [52]

Scenario 1						
Scenario Name		Normal sequence				
Use Case Step	**Triggering Event**	**Description Of Process**	**Information Exchanged**	**Producer**	**Receiver**	**Message Type**
1	Predetermined and/or variable polling frequency	D-SCADA on a predetermined and or variable frequency will poll the Distribution RTU of specified Feeder Devices	Poll of Distribution RTU	D-SCADA	Distribution RTU	Distribution Network Protocol (DNP3)
2		On a predetermined frequency Distribution RTU will Poll of specified Feeder Devices	Poll of specified Feeder Devices	Distribution RTU	Feeder Device	DNP3
...						
24		D-SCADA sends System Data to Distribution Historian on predetermined interval	System Data	D-SCADA	Distribution Historian	Proprietary

B.2 Use Case Description for BC2: Utility Operator Control Implementation

Table B.11: Use case name [19].

Use Case Identification		
ID	**Domain(s)**	**Name of Use Case**
BC2-UC1	Distribution, DER, Consumer	DG DemoNet—Smart LV Grid

Table B.13: Use case narrative [81].

Narrative of Use Case
Short description
The goal of the project was to find solutions for an active network operation for low voltage networks. Two of the main goals for the project were to find monitoring and control approaches in order to facilitate the system integration of DERs and electric mobility. The developed control concepts follow a step-by-step approach. In total, four stages were designed for the control concept, and implemented as a low voltage grid controller [81].
Complete description
The developed control concepts follow a step-by-step approach. In total, four stages were designed for the control concept, and implemented as a low voltage grid controller. The controller is an industrial PC and is located in the secondary substation, where it can access voltage measurements from smart meters in the field. The four control stages can be summarized as follows [81]: • *Stage 1—Local Control*: In this mode the local actuators only act on local measurements, and there is no communication between components. The main actuator is the MV/LV transformer equipped with an on-load-tap-changer. Additionally, PV inverters and EV charging stations are controlling the voltage locally using droop curves for reactive, Q(U), and active power, P(U). • *Stage 2—Distributed Control*: Here, a measurement and communication infrastructure is used. Voltage values from critical nodes in the grid are measured and transmitted to the central controller. The controller uses this information to find an optimal tap position of the transformer. Additionally the PV inverters and charging stations from Stage 1 are still in droop control mode. • *Stage 3—Coordinated Control*: Additionally Stage 1 and Stage 2 the coordinated control also updates the predefined Q(U) droop curves of the PV inverters and the active power droop curves charging stations. These updates are sent via broadcasts messages from the central controller. • *Stage 4—Selective Coordinated Control*: This stage is essentially the same as Stage 3 with the one exception. Updates of the droop curves are only sent to specific inverters and charging stations instead being broadcasted to all remote units.

Table B.15: Use case diagram [19].

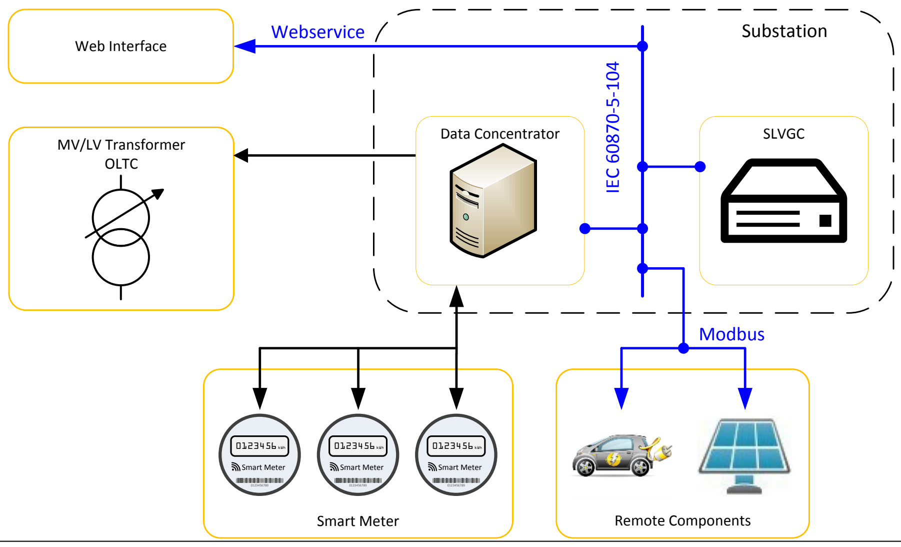
Diagram of Use Case


Table B.16: Use case actors [52].

Actors		
Actor Name	**Actor Type**	**Actor Description**
Web Interface	Application	Application to monitor and supervise the operation.
Tap Changer	Station Controller	Controller for the tap changing of the transformer.
Data Concentrator	Computer	Data concentrator for the smart meters.
SLVGC	Station Controller	Smart low voltage grid controller implementing the different control strategies.
PV Inverter	Distributed Energy Resource	Responsible for local voltage optimization using volt-VAr control.
Smart Meters	End Point Monitor	Voltage monitors for endpoints and critical points of the line.

Table B.18: Scenario conditions.

Scenario Conditions			
No.	**Scenario Name**	**Primary Actor**	**Triggering Event**
1	Stage 1—Local Control	PV Inverter	Continuous
2	Stage 2—Distributed Control	SLVGC	Continuous
3	Stage 3—Coordinated Control	SLVGC	Continuous
4	Stage 4—Selective Coordinated Control	SLVGC	Continuous

Table B.20: Excerpt of steps for Scenario 3.

Scenario 3					
Scenario Name		Stage 3—Coordinated Control			
Step No.	**Event**	**Description of Process**	**Information Producer**	**Information Receiver**	**Information Exchanged**
1	Time trigger	The Data Concentrator requests voltage measurement from smart meters.	Data Concentrator	Smart Meters	IEX-1
2		The smart meters returns voltage measurement to the Data Concentrator.	Smart Meters	Data Concentrator	IEX-2
...					
10		The SLVGC calculates the voltage spread.			
11	Voltage spread above limit	The SLVGC calculates new curve setpoints for the PV Inverter			
12		The new curve setpoints are sent to the PV Inverter.	SLVGC	PV Inverter	IEX-3
...					
20	Voltage spread below limit; voltage too high	The SLVGC calculates new tap position for the Tap Changer	SLVGC	Tap Changer	IEX-4
...					

Table B.22: Information exchanged.

Information Exchanged		
Information Exchanged ID	**Name of Information Exchanged**	**Description of Information Exchanged**
IEX-1	Voltage measurement request	Request for the current voltage measurement
IEX-2	Voltage measurement	Voltage measurement message as response of a request
IEX-3	New volt-VAr curve points	The new calculated volt-VAr curve points
IEX-4	Tap postition	The tap position for the transformer

B.3 Use Case Description for BC3: Ancillary Services from Component Manufacturers

Table B.24: Use case name.

Use Case Identification		
ID	**Domain(s)**	**Name of Use Case**
BC3-UC1	DER	Q(U) control of DER

Table B.26: Use case narrative [157].

Narrative of Use Case
Short description
The Q(U) characteristic defines the reactive power provision of the DER depending on the voltage measured at the point of coupling.
Complete description
With an active power output, either a fixed reactive power provision is used or the network operator can adjust the target value through remote control. One possibility is that the reactive power setting is adjusted through a reactive power/voltage characteristic Q(U). For the Q(U) characteristic it should be possible for the network operator to adjust the reaction time between 10 s and 1 min. Furthermore, to avoid voltage jumps it is advisable to choose a characteristic with continuous profile and limited gradient. The Q(U) characteristic defines the reactive power provision depending on the voltage measured at the point of coupling of the DER.

Table B.28: Use case diagram.

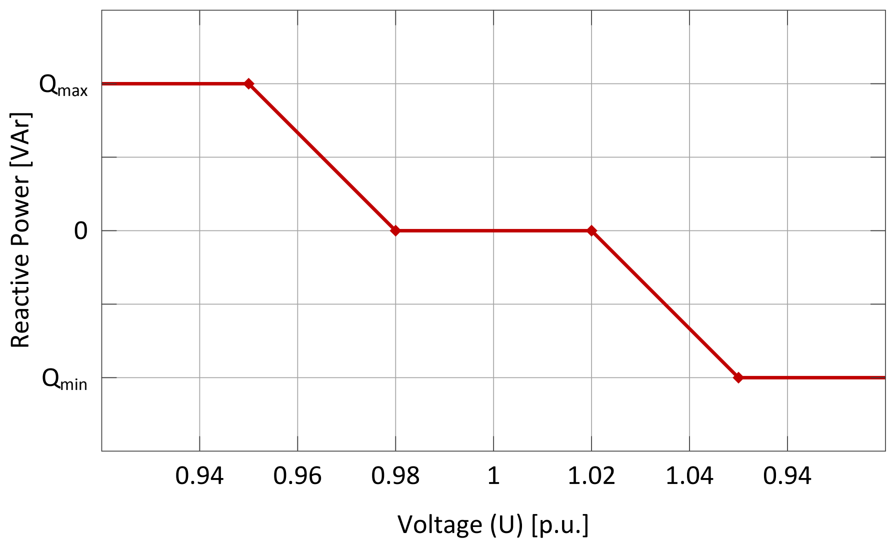
Diagram of Use Case


Table B.29: Use case actors.

Actors		
Actor Name	**Actor Type**	**Actor Description**
Q(U) Controller	Controller	Microprocessor-based controller with a programmable memory for the internal storage of user-defined instructions. It is part of the DER and is responsible for the Q(U) control.
DER	Distributed Energy Resource	Small unit which generates energy and which is connected to the distribution grid through an inverter. It implements the reactive power setpoints from the Q(U) Controller.
Voltage Meter	End Point Monitor	Voltage monitor for the point of coupling of the DER.

Table B.31: Scenario conditions.

Scenario Conditions			
No.	**Scenario Name**	**Primary Actor**	**Triggering Event**
5	DER volt-VAr control	Q(U) Controller	Continuous

Table B.33: Steps for normal operation.

Scenario 5					
Scenario Name		DER volt-VAr control			
Step No.	**Event**	**Description of Process**	**Information Producer**	**Information Receiver**	**Information Exchanged**
1	Time trigger	The Q(U) Controller requests current voltage level from the Voltage Meter.	Q(U) Controller	Voltage Meter	IEX-1
2		The Voltage Meter returns voltage measurement to the Q(U) Controller.	Voltage Meter	Q(U) Controller	IEX-2
3		The Q(U) Controller calculates a new reactive power setpoint based on its current volt-VAr curve.			
4		The new reactive power setpoint is sent to the DER	Q(U) Controller	DER	IEX-3
5	Receives reactive power setpoint	The DER uses the new reactive power setpoint.			

Table B.35: Information exchanged.

Information Exchanged		
Information Exchanged ID	**Name of Information Exchanged**	**Description of Information Exchanged**
IEX-1	Voltage measurement request	Request for the current voltage measurement
IEX-2	Voltage measurement	Voltage measurement message as response of a request
IEX-3	Reactive power setpoint	The reactive power setpoint for the inverter

B.4 Test Case Description

Table B.37: Use case identification.

Use Case Identification		
ID	**Domain(s)**	**Name of Use Case**
TC1	Distribution, DER	Coordinated Volt-VAr Control

Table B.39: Version management.

Version Management				
Version No.	**Date**	**Name of author(s)**	**Changes**	**Approval Status**
1.0	2018-01-24	Filip Pröstl Andrén	Initial version	First draft

Table B.41: Scope and objectives of use case.

Scope and Objectives of Use Case	
Scope	Voltage optimization using local and central control
Objective	Keeping voltage within the defined limits

Table B.43: Narrative of use case.

Narrative of Use Case
Short description
To keep the voltage in the network within the allowed limits local volt-VAr control is provided by selected DERs throughout the network. Additionally, at the operation level, a central volt-VAr optimizer supervises and manages the local DER controllers in order to achieve an optimal power level within the whole distribution network.
Complete description

The scenario of the use case is typical for distribution network operation. The primary goal is to keep the voltage in the network within the allowed limits. To achieve this local voltage control is provided by selected DERs throughout the network. These DERs affect the voltage at their point of coupling through the use of volt-VAr control [71]. Additionally, at the operation level, a central volt-VAr optimizer supervises and manages the local DER controllers in order to achieve an optimal power level within the whole distribution network. The devices in the field (i.e., the *Transformer Monitor*, the *Bus Monitor*, the *DER Controller*, and the *End-of-Line Monitor*) send measurements to the *Distribution RTU*. The *Distribution RTU* aggregates this information and forwards it to the *D-SCADA* system. Thereupon, the *D-SCADA* forwards all vital information to the *Volt-VAr Controller*. It computes new volt-VAr set points for the *DER Controllers*. The new set points are forwarded by the *D-SCADA* and the *Distribution RTU* to the field controllers. With the volt-VAr set points the *DER Controllers* can calculate a reactive power set point for the *DER Generator*.

Table B.45: Diagram of use case.

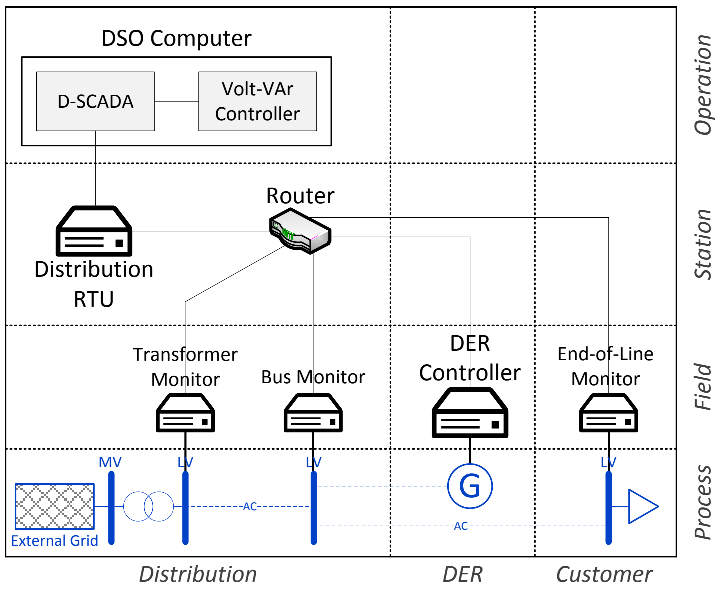
Diagram of Use Case
 DSO Computer D-SCADA Volt-VAr Controller Router Distribution RTU Transformer Monitor Bus Monitor DER Controller End-of-Line Monitor MV LV LV LV AC AC G External Grid Operation Station Field Process Distribution DER Customer

Table B.46: Use case actors.

Actors		
Actor Name	**Actor Type**	**Actor Description**
Volt-VAr Controller	Station Controller	Controller located in the substation monitoring and controlling the devices in the network. It optimizes the voltage level in the grid by issuing new reactive power set points to the field components.

D-SCADA	SCADA	The SCADA application provides the basic functionality for managing the grid, especially provides the communication with the substations to monitor and control the grid.
Distribution RTU	IED	Incorporates one or more processors with the capability to receive or send data/control between the field devices and the DSO station controller.
DER Controller	Controller	Microprocessor-based controller with a programmable memory for the internal storage of user-defined instructions. It is responsible for the high-level controls, including the local volt-VAr control, of the DER Generator.
DER Generator	Distributed Energy Resource	Small unit which generates energy and which is connected to the distribution grid through an inverter. It implements the reactive power setpoints from the DER Controller.
Transformer Monitor	End Point Monitor	A monitor of electricity not used for billing purposes and deployed by the DSO for the purposes of LV visibility of per-premises consumption.
Bus Monitor	End Point Monitor	A monitor of electricity not used for billing purposes and deployed by the DSO for the purposes of LV visibility of per-premises consumption.
End-of-Line Monitor	End Point Monitor	A monitor of electricity not used for billing purposes and deployed by the DSO for the purposes of LV visibility of per-premises consumption.

Table B.48: Scenario conditions.

Scenario Conditions			
No.	**Scenario Name**	**Primary Actor**	**Triggering Event**
6	Distribution system voltage control	Volt-VAr Controller	Voltage spread too high
7	DER volt-VAr control	DER Controller	Continuous

Table B.50: Steps for Scenario 6.

Scenario 6					
Scenario Name	Distribution system voltage control				
Step No.	**Event**	**Description of Process**	**Information Producer**	**Information Receiver**	**Information Exchanged**
1	Time trigger	The Distribution RTU requests voltage measurement from grid monitors.	Distribution RTU	Transformer Monitor	IEX-1

2	Time trigger	The Distribution RTU requests voltage measurement from grid monitors.	Distribution RTU	Bus Monitor	IEX-1
3	Time trigger	The Distribution RTU requests voltage measurement from grid monitors.	Distribution RTU	End-of-Line Monitor	IEX-1
4	Time trigger	The Distribution RTU requests voltage measurement from DER Controller.	Distribution RTU	DER Controller	IEX-1
5		The Transformer Monitor returns voltage measurement to Distribution RTU.	Transformer Monitor	Distribution RTU	IEX-2
6		The Bus Monitor returns voltage measurement to Distribution RTU.	Bus Monitor	Distribution RTU	IEX-2
7		The End-of-Line Monitor returns voltage measurement to Distribution RTU.	End-of-Line Monitor	Distribution RTU	IEX-2
8		The DER Controller returns voltage measurement to Distribution RTU.	DER Controller	Distribution RTU	IEX-2
9		The Distribution RTU forwards collected measurements to the D-SCADA.	Distribution RTU	D-SCADA	IEX-3
10		The D-SCADA forwards measurements to the Volt-VAr Controller.	D-SCADA	Volt-VAr Controller	IEX-3
11		The Volt-VAr Controller calculates the voltage spread.			
12	Voltage spread above limit	The Volt-VAr Controller requests the current volt-VAr curve points from the DER Controller.	Volt-VAr Controller	D-SCADA	IEX-4
13		The D-SCADA forwards the requests.	D-SCADA	Distribution RTU	IEX-4
14		The Distribution RTU forwards the requests.	Distribution RTU	DER Controller	IEX-4
15		The DER Controller returns its current curve points.	DER Controller	Distribution RTU	IEX-5
16		The Distribution RTU forwards the current curve points.	Distribution RTU	D-SCADA	IEX-5
17		The D-SCADA forwards the current curve points.	D-SCADA	Volt-VAr Controller	IEX-5
18		The Volt-VAr Controller calculates new curve setpoints for the DER Controller			
19		The new curve setpoints are sent to the DER Controller.	Volt-VAr Controller	D-SCADA	IEX-6

20		The new curve setpoints are forwarded.	D-SCADA	Distribution RTU	IEX-6
21		The new curve setpoints are forwarded.	Distribution RTU	DER Controller	IEX-6
22	Receives curve points	The DER Controller uses the new volt-VAr curve points			

Table B.52: Steps for Scenario 7.

Scenario 7					
Scenario Name		DER volt-VAr control			
Step No.	**Event**	**Description of Process**	**Information Producer**	**Information Receiver**	**Information Exchanged**
1	Time trigger	The DER Controller requests current voltage level from the DER Generator.	DER Controller	DER Generator	IEX-1
2		The DER Generator returns voltage measurement to DER Controller.	DER Generator	DER Controller	IEX-2
3		The DER Controller calculates a new reactive power setpoint based on its current volt-VAr curve.			
4		The new reactive power setpoint is sent to the DER Generator	DER Controller	DER Generator	IEX-7
5	Receives reactive power setpoint	The DER Generator uses the new reactive power setpoint.			

Table B.54: Inforamtion exchanged.

Information Exchanged		
Information Exchanged ID	**Name of Information Exchanged**	**Description of Information Exchanged**
IEX-1	Voltage measurement request	Request for the current voltage measurement
IEX-2	Voltage measurement	Voltage measurement message as response of a request
IEX-3	Aggregated voltage measurement	Aggregated voltage measurements from all field components
IEX-4	Volt-VAr curve request	Request for the currently used volt-VAr curve points
IEX-5	Volt-VAr curve points	The currently used volt-VAr curve points
IEX-6	New volt-VAr curve points	The new calculated volt-VAr curve points

IEX-7	Reactive power setpoint	The reactive power setpoint for the inverter

APPENDIX C

Use Case Listings

C.1 Application Specification for the Test Case

```
application VoltVArControlCoordinated {
  /* Volt-VAr Controller:
   * Controller located in the substation monitoring and controlling the devices
   * in the network. It optimizes the voltage level in the grid by issuing new
   * reactive power set points to the field components.
   */
  @Distribution @Operation
  function VoltVArController at DistributionSystemVV.DSOComputer.VoltVAr {
    consumes Measurements.AggregatedMeasurement gridStatus
    requests DERCtrlInterfaces.DERVoltVArCurve dscadaVoltVArCurve
  }

  /* D-SCADA:
   * The SCADA application provides the basic functionality for managing the grid,
   * especially provides the communication with the substations to monitor and
   * control the grid.
   */
  @Distribution @Station
  function DSCADA at DistributionSystemVV.DSOComputer.SCADA {
    emits Measurements.AggregatedMeasurement gridStatus
    provides DERCtrlInterfaces.DERVoltVArCurve voltVArCurve
    consumes FieldInformation.GridStatus gridStatusRTU
    requests DERCtrlInterfaces.DERVoltVArCurve rtuVoltVArCurve
  }

  /* Distribution RTU:
   * Incorporates one or more processors with the capability to receive or send
   * data/control between the field devices and the DSO station controller.
   */
  @Distribution @Station
  function DistributionRTU at DistributionSystemVV.DistributionRTU.RTUResource {
```

```
  emits Measurements.AggregatedMeasurement gridStatus
  provides DERCtrlInterfaces.DERVoltVArCurve voltVArCurve
  requests DERCtrlInterfaces.DERVoltVArCurve derVoltVArCurve
  requests Measurements.GridMeasurements derMeasurements
  requests Measurements.GridMeasurements transformerMeasurements
  requests Measurements.GridMeasurements busMeasurements
  requests Measurements.GridMeasurements eolMeasurements
}

/* DER Controller:
 * Microprocessor-based controller with a programmable memory for the internal
 * storage of user-defined instructions. It is responsible for the high-level
 * controls, including the local volt-VAr control, of the DER Generator.
 */
@DER @Field
function DERController at DistributionSystemVV.DERController.AncillaryServices {
  provides DERCtrlInterfaces.DERVoltVArCurve voltVar
  provides Measurements.GridMeasurements measurements
  requests DERCtrlInterfaces.DERDirectControls derDirectControls
  requests Measurements.GridMeasurements derMeasurements
}

/* DER Generator:
 * Small unit which generates energy and which is connected to the distribution
 * grid through an inverter. It implements the reactive power setpoints from the
 * DER Controller.
 */
@DER @Process
function DERGenerator at DistributionSystemVV.DERGenerator.DERResource {
  provides DERCtrlInterfaces.DERDirectControls directControls
  provides Measurements.GridMeasurements measurements
}

/* Transformer Monitor:
 * A monitor of electricity not used for billing purposes and deployed by the DSO
 * for the purposes of LV visibility of per-premises consumption.
 *
 */
@Distribution @Field
function TransformerMonitor at ⤶
    ↳ DistributionSystemVV.TransformerMonitor.MonitorResource {
  provides Measurements.GridMeasurements measurements
}

/* Bus Monitor:
 * A monitor of electricity not used for billing purposes and deployed by the DSO
 * for the purposes of LV visibility of per-premises consumption.
 *
 */
@Distribution @Field
function BusMonitor at DistributionSystemVV.BusMonitor.MonitorResource {
  provides Measurements.GridMeasurements measurements
}
```

```

/* End-of-Line Monitor:
 * A monitor of electricity not used for billing purposes and deployed by the DSO
 * for the purposes of LV visibility of per-premises consumption.
 *
 */
@Customer Field
function EndOfLineMonitor at DistributionSystemVV.EndOfLineMonitor.MonitorResource {
  provides Measurements.GridMeasurements measurements
}

/* Module for measurements */
module Measurements {
  /* Interface for measurements at a single point in the grid */
  interface GridMeasurement {
      /* Arithmetic average of the phase to phase voltage for 3 phases */
      readonly attribute float32 voltage
  }
  /* Event for aggregated voltage measurements */
  eventtype AggregatedMeasurement {
      /* Aggregated measurements from field devices (p.u.) */
    public float32 vTM, vBM, vEOLM, vDER
}}

/* Module for DER controls */
module DERCtrlInterfaces {
  /* Representation of the volt-VAr curve */
  interface DERVoltVArCurve {
    /* Voltage set points for droop curve */
    attribute float32 v1, v2, v3, v4
    /* Reactive power set points for droop curve */
    attribute float32 q1, q2, q3, q4
  }

  /* Interface for direct controls of the DER Generator */
  interface DERDirectControls {
    attribute float32 qSetPoint
}}

/* Connections between service implementations */
connect DERController.derDirectControls with DERGenerator.directControls
connect DERController.derMeasurements with DERGenerator.measurements
connect DistributionRTU.derVoltVArCurve with DERController.voltVar
connect DistributionRTU.derMeasurements with DERController.measurements
connect DistributionRTU.transformerMeasurements with TransformerMonitor.measurements
connect DistributionRTU.busMeasurements with BusMonitor.measurements
connect DistributionRTU.eolMeasurements with EndOfLineMonitor.measurements
connect DSCADA.gridStatusRTU with DistributionRTU.gridStatus
connect DSCADA.rtuVoltVArCurve with DistributionRTU.voltVArCurve
connect VoltVArController.gridStatus with DSCADA.gridStatus
connect VoltVArController.dscadaVoltVArCurve with DSCADA.voltVArCurve
}
```

Listing C.1: Full business case and functional specification for the example use case.

C.2 System Specification for the Test Case

```
system DistributionSystemVV {
  /* ICT components */

  /* DSO Computer:
   * The computer at the DSO wich contains the processes for the D-SCADA and the
   * Volt-VAr Controller processes.
   */
  @Distribution @Operation
  device DSOComputer {
    ethernet eth0 {ip = "10.0.0.1"}
    resource SCADA
    resource VoltVAr
  }
  /* Distribution RTU:
   * The RTU at the station zone. It contains the process for the DistributionRTU
   * process.
   */
  @Distribution @Station
  device DistributionRTU {
    ethernet eth0 {ip = "10.0.0.2"}
    ethernet eth0 {ip = "101.0.0.1"}
    resource RTUResource
  }
  /* Station router
   * Ethernet router between DSO network and field network.
   */
  @Distribution @Station
  router StationRouter
  /* Transformer Montior:
   * Measurement device which monitors the voltage at the transformer.
   */
  @Distribution @Field
  device TransformerMonitor {
    ethernet eth0 {ip = "101.0.0.2"}
    resource MonitorResource
  }
  /* Bus Montior:
   * Measurement device which monitors the voltage at the LV bus.
   */
  @Distribution @Field
  device BusMonitor {
    ethernet eth0 {ip = "101.0.0.3"}
    resource MonitorResource
  }
  /* DER Controller:
   * An embedded controller connected with the DER Generator.
   */
  @DER @Field
  device DERController {
    ethernet eth0 {ip = "101.0.0.4"}
    ethernet eth0 {ip = "192.168.0.2"}
    resource AncillaryServices
  }
  /* End-Of-Line Montior:
   * Measurement device which monitors the voltage at the end of the line.
   */
  @Customer @Field
  device EndOfLineMonitor {
    ethernet eth0 {ip = "101.0.0.5"}
    resource MonitorResource
  }

  /* Electrical components */

  /* External Grid:
   * The external grid represents the MV grid to which the use case grid is
   * connected.
   */
  @Distribution @Process
  generator ExternalSystem {
    terminal MVBus
  }
  /*
   * The MV bus above the transformer
   */
  @Distribution @Process
  busbar MVBus {
    terminal ExternalSystem
    terminal MV2LVTransformer
  }
  /*
   * The MV to LV transformer, contining two windings
   */
  @Distribution @Process
  transformer MV2LVTransformer {
    winding MV {
      terminal mvSide
    }
    winding LV {
      terminal lvSide
  }}
  /*
   * The LV busbar after the transformer
   */
  @Distribution @Process
  busbar LVBus1 {
    terminal MV2LVTransformer
    terminal Line1
  }
  /*
   * The line between the first two LV busbars
   */
  line Line1 {
    terminal LVBus1
    terminal LVBus2
  }
  /*
   * The second LV busbar
   */
  @Distribution @Process
  busbar LVBus2 {
    terminal Line1
    terminal DERGenerator
    terminal Line2
  }
  /* DER Generator:
   * It contains an Ethernet interface as well as a computational resource.
   */
  @DER @Process
  generator DERGenerator {
    ethernet eth0 {ip = "192.168.0.1"}
    terminal LVBus2
    resource DERResource
  }
  /*
   * The line between the second and the third LV busbars
   */
  line Line2 {
    terminal LVBus2
    terminal LVBus3
  }
  /*
   * The third LV busbars
   */
  @Customer @Process
  busbar LVBus3 {
    terminal Line2
    terminal Load
  }
  /*
   * The load at the end of the line
   */
  @Customer @Process
  consumer Load {
    terminal LVBus3
  }

  /* Connections between the ICT components */
  connect DistributionRTU.eth0 with DSOComputer.eth0
  connect DistributionRTU.eth1 with StationRouter
  connect TransformerMonitor.eth0 with StationRouter
  connect BusMonitor.eth0 with StationRouter
  connect DERController.eth0 with StationRouter
  connect DERController.eth1 with DERGenerator.eth0
  connect EndOfLineMonitor.eth0 with StationRouter

  /* Connections between electrical components */
```

```
  connect MVBus.ExternalSystem with ExternalSystem.MVBus
  connect MV2LVTransformer.MV.mvSide with MVBus.MV2LVTransformer
  connect LVBus1.MV2LVTransformer with MV2LVTransformer.LV.lvSide
  connect Line1.LVBus1 with LVBus1.Line1
  connect LVBus2.Line1 with Line1.LVBus2
  connect DERGenerator.LVBus2 with LVBus2.DERGenerator
  connect Line2.LVBus2 with LVBus2.Line2
  connect LVBus3.Line2 with Line2.LVBus3
  connect Load.LVBus3 with LVBus3.Load
}
```

Listing C.2: Full system specification for the example use case.

Acronyms

AADL Architecture Analysis and Design Language. 20, 69

AC Alternating Current. 57, 133

ADL Architecture Description Language. 20

API Application Programming Interface. 26, 32, 97

ASN.1 Abstract Syntax Notation One. 24, 102, 103, 129

ATL ATL Transformation Language. 36, 100, 101, 103, 104

AUTOSAR AUTomotive Open System ARchitecture. 20, 39

BDA Basic Data Attribute. 88

CBSE Component-Based Software Engineering. 37, 39

CDC Common Data Class. 28, 30, 87

CHIL Controller Hardware-In-the-Loop. 33, 34

CI Continuous Integration. 142, 143

CIM Common Information Model. 19, 20, 23, 27, 32, 40, 66–69, 75, 76, 92, 96, 104, 118, 128, 129, 139, 148

COM Component Object Model. 39

CORBA Common Object Request Broker Architecture. 39

COSEM Companion Specification for Energy Metering. 40

CPES Cyber-Physical Energy System. 65

D-SCADA DSO SCADA. 106, 160, 161, 168

DA Data Attribute. 29, 30, 87

DC Direct Current. 57, 133

DER Distributed Energy Resource. 2–4, 32, 46–55, 65, 68, 105–107, 109, 118, 121, 122, 125, 126, 128, 138, 160, 162, 165–168

DG Distributed Generation. 51

DGEN Logical Node: DER unit generator. 69

DLL Dynamic Link Library. 129, 130

DLMS Distribution Line Message Service. 28

DM Domain Model. 65–70, 75, 77, 79, 82, 138

DMS Distribution Management Systems. 19, 27

DNP3 Distribution Network Protocol. 161

DO Data Object. 28, 87

DRCC Logical Node: DER supervisory control. 118, 119

DSL DIgSILENT Simulation Language. 129

DSL Domain-Specific Language. 18, 35, 37, 71, 99, 147

DSO Distribution System Operator. xv, 46, 47, 55, 56, 105, 106, 121, 124, 126, 130, 131, 138, 160, 169, 181

EBNF Extended Backus–Naur Form. 74, 155

EMS Energy Management System. 19, 27

EPRI Electric Power Research Institute. 15, 19, 45

EV Electric Vehicle. 2, 28, 49, 162

FB Function Block. 23–25, 40, 81, 82, 98, 140, 141

FBD Function Block Diagram. 21

FMAR Logical Node: Establish mode curves and parameters. 118, 119

GOOSE Generic Object Oriented Substation Event. 29

GSE Generic Substation Event. 29

HIL Hardware-In-the-Loop. 33, 34

HuT Hardware-under-Test. 33, 34

ICT Information and Communication Technology. 3, 4, 15, 18, 20, 21, 26–28, 32–34, 49, 50, 52, 57, 58, 61, 62, 65, 66, 71–73, 75, 77, 111, 119, 121, 132

IDE Integrated Development Environment. 99–101

IDL Interface Description Language. 38, 67, 77, 78, 138

IEC International Electrotechnical Commission. 19, 27, 30

IED Intelligent Electronic Device. 28, 29, 47, 48, 50, 87, 94, 97, 121

IL Instruction List. 21

ILIM Implementation Language Independent Model. 98

IP Internet Protocol. 71, 72, 94, 96, 111, 129, 133

LD Ladder Diagram. 21, 28, 29

LN Logical Node. 28–30, 35, 67, 69, 87, 88, 90, 94, 103, 118, 140

LV Low Voltage. 3, 49, 50, 112, 133, 162, 169

M2M model-to-model. 36

M2T model-to-text. 36

MAC Media Access Control. 71, 72

MDA Model-Driven Architecture. 34, 37, 39, 69, 97, 98

MDE Model-Driven Engineering. 6, 11, 34–37, 39–41, 61, 69, 82, 97, 138, 139, 144, 145, 147, 149–151

MMS Manufacturing Message Specification. 29

MMXU Logical Node: Measurement. 87, 88, 90, 100, 118, 119

MOF Meta Object Facility. 35, 37

MOFM2T MOF Model to Text. 36

MQTT Message Queue Telemetry Transport. 102

MV Medium Voltage. 3, 49, 50, 53, 111, 112, 162

NIST National Institute of Standards and Technology. 39

OMG Object Management Group. 37, 77, 78, 138

OPC UA OPC Unified Architecture. 28, 47, 87, 102

OSGi Open Services Gateway initiative. 39

OSI Open Systems Interconnection. 71, 119

P Active Power. 49, 162

p.u. per unit. 134

PC Personal Computer. 49, 50, 162

PHIL Power Hardware-In-the-Loop. 33, 34

PIM Platform-Independent Model. 37, 39, 40, 69, 97, 98

PLC Power Line Carrier. 21, 22, 49, 66

PSAL Power System Automation Language. 71–78, 80–82, 84–89, 94–97, 99–101, 103, 104, 107, 108, 110, 119–121, 128, 138, 140–144, 146–148, 150, 155

PSM Platform-Specific Model. 37, 39, 40, 69, 98, 138

PV Photovoltaic. 2, 46, 49–51, 133, 162

Q Reactive Power. 49, 121, 162

QVT Query View Transformation. 36, 37

RDF Resource Description Framework. 20

RTE Runtime Environment. 102

RTU Remote Terminal Unit. 106, 160, 161, 168

SCADA Supervisory Control and Data Acquisition. 19, 106, 159, 169, 181

SCL System Configuration Language. 29, 40, 66, 87, 88, 94, 95, 97, 98, 100, 103, 104, 133, 134, 139, 141, 143, 144, 148

SDN Software Defined Networking. 47, 94

SFC Sequential Function Chart. 21

SGAM Smart Grid Architecture Model. 5, 17–19, 22, 23, 39, 44, 45, 57, 61, 63, 65, 66, 70–73, 107, 118, 138–140, 142, 143, 147, 148

SIFB Service Interface Function Block. 25, 82, 83, 92–94, 98, 102, 103, 129, 130

SLVGC Smart Low Voltage Grid Controller. 50

SOA Service-Oriented Architecture. 28

ST Structured Text. 21

SysML System Modeling Language. 20, 69

T2M text-to-model. 36

TC Technical Committee. 27, 30

TCP Transmission Control Protocol. 66, 102, 129, 133

TSO Transmission System Operator. 55

U Voltage. 49, 121, 162

UCMR Use Case Management Repository. 16

UDP User Datagram Protocol. 66, 102

UML Unified Modeling Language. 16, 18–20, 30, 35, 37, 40, 57, 66, 69, 72, 73, 142

VFB Virtual Functional Bus. 121

VLAN Virtual Local Area Network. 71, 94

VVC Volt-VAr Controller. 45, 160

XML Extensible Markup Language. 29, 87, 96, 98

Bibliography

[1] "Acceleo," Jun. 2017. [Online]. Available: https://www.eclipse.org/acceleo/

[2] D. J. Anderson and A. Carmichael, *Essential Kanban Condensed.* Blue Hole Press, Nov. 2015.

[3] F. Andrén, R. Bründlinger, and T. Strasser, "IEC 61850/61499 Control of Distributed Energy Resources: Concept, Guidelines, and Implementation," *IEEE Transactions on Energy Conversion*, vol. 29, no. 4, pp. 1008–1017, Dec. 2014.

[4] F. Andrén, F. Lehfuss, and T. Strasser, "A Development and Validation Environment for Real-Time Controller-Hardware-in-the-Loop Experiments in Smart Grids," *International Journal of Distributed Energy Resources and Smart Grids*, vol. 9, no. 1, pp. 27–50, 2013.

[5] F. Andrén, M. Stifter, and T. Strasser, "Towards a Semantic Driven Framework for Smart Grid Applications: Model-Driven Development using CIM, IEC 61850 and IEC 61499," *Informatik-Spektrum*, vol. 36, no. 1, pp. 58–68, Jan. 2013.

[6] F. Andrén, T. Strasser, and W. Kastner, "Model-driven engineering applied to Smart Grid automation using IEC 61850 and IEC 61499," in *Power Systems Computation Conference (PSCC)*, Aug. 2014, pp. 1–7.

[7] F. Andrén, T. Strasser, A. Zoitl, and I. Hegny, "A reconfigurable communication gateway for distributed embedded control systems," in *38th Annual Conf. of the IEEE Ind. Electronics Society (IECON)*, 2012.

[8] F. Andrén, T. Strasser, and W. Kastner, "Towards a Common Modeling Approach for Smart Grid Automation," in *39th Annual Conference on IEEE Industrial Electronics Society (IECON)*, 2013, pp. 5338–5344.

[9] F. Andrén, T. Strasser, S. Rohjans, and M. Uslar, "Analyzing the need for a common modeling language for Smart Grid applications," in *11th IEEE International Conference on Industrial Informatics (INDIN)*, 2013.

[10] Angelo Frascella et al., "Looking for the unified classification and evaluation approach of SG interface standards for the purposes of ELECTRA IRP," in *Int. Symp. on Smart Electric Distribution Systems and Technologies (EDST)*, 2015.

[11] L. Ardito, G. Procaccianti, G. Menga, and M. Morisio, "Smart Grid Technologies in Europe: An Overview," *Energies*, vol. 6, no. 1, pp. 251–281, Jan. 2013.

[12] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: advanced concepts and tools for in-place EMF model transformations," *Model Driven Engineering Languages and Systems*, pp. 121–135, 2010.

[13] J. Bastian, C. Claus, S. Wolf, and P. Schneider, "Master for co-simulation using FMI," in *8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany.* Linköping University Electronic Press, 2011, pp. 115–120.

[14] B. Boehm, "Get ready for agile methods, with care," *Computer*, vol. 35, no. 1, pp. 64–69, 2002.

[15] B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed, Portable Documents.* Addison-Wesley Professional, 2003.

[16] D. Box, *Essential COM.* Addison-Wesley Professional, 1998.

[17] P. Brédillet, E. Lambert, and E. Schultz, "CIM, 61850, COSEM Standards Used in a Model Driven Integration Approach to Build the Smart Grid Service Oriented Architecture," in *2010 First IEEE International Conference on Smart Grid Communications (SmartGridComm)*, Oct. 2010, pp. 467–471.

[18] C. Brunner, "IEC 61850 for power system communication," in *Transmission and Distribution Conference and Exposition, 2008. T&D. IEEE/PES.* IEEE, 2008, pp. 1–6.

[19] H. Brunner, "DG DemoNet - Smart LV Grid," 2015.

[20] A. Caragliu, C. Del Bo, and P. Nijkamp, "Smart cities in Europe," *Journal of urban technology*, vol. 18, no. 2, pp. 65–82, 2011.

[21] C. Cecati, C. Citro, A. Piccolo, and P. Siano, "Smart Operation of Wind Turbines and Diesel Generators According to Economic Criteria," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 10, pp. 4514–4525, 2011.

[22] C. Cecati, G. Hancke, P. Palensky, P. Siano, and X. Yu, "Guest Editorial Special Section on Information Technologies in Smart Grids," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1380–1383, Aug. 2013.

[23] "CEI 0-21 Reference technical rules for the connection of active and passive users to the LV electrical utilities," CEI, Tech. Rep., 2012.

[24] CEN-CENELEC-ETSI Smart Grid Coordination Group, "SG-CG/M490/F Overview of SG-CG Methodologies," Tech. Rep., 2014.

[25] CEN-CENELEC-ETSI Smart Grid Working Group Reference Architecture, “Reference Architecture for the Smart Grid,” Tech. Rep., 2012.

[26] CEN-CENELEC-ETSI Smart Grid Working Group Sustainable Processes, “Use Case Collection, Management, Repository, Analysis and Harmonization,” Tech. Rep., 2012.

[27] “CET850 IEC 61850 configuration tool V2.1 | Schneider Electric,” Nov. 2017. [Online]. Available: https://www.schneider-electric.us/en/download/document/CET850+IEC850+configuration+tool/

[28] “Common Object Request Broker Architecture,” Object Management Group (OMG), Tech. Rep., 2012. [Online]. Available: http://www.omg.org/spec/CORBA/3.3/

[29] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. V. Chaudron, “A Classification Framework for Software Component Models,” *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 593–615, Sep. 2011.

[30] K. Czarnecki and S. Helsen, “Feature-based survey of model transformation approaches,” *IBM Systems Journal*, vol. 45, no. 3, pp. 621–645, 2006.

[31] C. Dänekas, C. Neureiter, S. Rohjans, M. Uslar, and D. Engel, “Towards a Model-Driven-Architecture Process for Smart Grid Projects,” in *Digital Enterprise Design & Management*, ser. Advances in Intelligent Systems and Computing, P. J. Benghozi, D. Krob, A. Lonjon, and H. Panetto, Eds. Springer International Publishing, 2014, vol. 261, pp. 47–58.

[32] K. Dyke, N. Schofield, and M. Barnes, “The Impact of Transport Electrification on Electrical Networks,” *IEEE Transactions on Industrial Electronics*, vol. 57, no. 12, pp. 3917–3926, 2010.

[33] “EBNF for XML,” World Wide Web Consortium (W3C), Tech. Rep., 2004. [Online]. Available: https://www.w3.org/TR/2004/REC-xml-20040204/#sec-notation

[34] EC, “M/490 Standardization Mandate to European Standardisation Organisations (ESOs) to support European Smart Grid deployment,” European Commission (EC), Tech. Rep., 2012.

[35] “Eclipse - The Eclipse Foundation open source community website.” [Online]. Available: http://www.eclipse.org

[36] S. Efftinge and M. Völter, “oAW xText: A framework for textual DSLs,” in *Workshop on Modeling Symposium at Eclipse Summit*, vol. 32, 2006.

[37] G. Elliott, *Global business information technology: an integrated systems approach.* Pearson Education, 2004.

[38] "EPRI SmartGrid Resource Center - Use Case Repository," Feb. 2017. [Online]. Available: http://smartgrid.epri.com/Repository/Repository.aspx

[39] T. Facchinetti and M. L. Della Vedova, "Real-time modeling for direct load control in cyber-physical power systems," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 689–698, 2011.

[40] H. Farhangi, "The path of the smart grid," *IEEE Power and Energy Magazine*, vol. 8, no. 1, pp. 18–28, 2010.

[41] M. Faschang, "Rapid control prototyping for networked Smart Grid systems based on an agile development process," Ph.D. dissertation, Vienna University of Technology, 2015.

[42] P. H. Feiler and D. P. Gluch, *Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language.* Addison-Wesley, 2012.

[43] E. Ferreira, R. Paulo, and P. Henriques, "Integration of the ST language in a model-based engineering environment for control systems: An approach for compiler implementation," *Computer Science and Information Systems*, vol. 5, no. 2, pp. 87–101, 2008.

[44] M. Fleck, "Search-Based Model Transformations," Ph.D. dissertation, Institute of Software Technology and Interactive Systems, Vienna University of Technology, 2016.

[45] M. Fleck, J. Troya, and M. Wimmer, "Search-based model transformations," *Journal of Software: Evolution and Process*, vol. 28, no. 12, pp. 1081–1117, Dec. 2016.

[46] K. Forsberg and H. Mooz, "The Relationship of Systems Engineering to the Project Cycle," *Engineering Management Journal*, vol. 4, no. 3, pp. 36–43, Sep. 1992.

[47] M. Fowler, *Domain-specific languages.* Pearson Education, 2010.

[48] M. Fowler and J. Highsmith, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[49] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange, "AUTOSAR–A Worldwide Standard is on the Road," in *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, vol. 62, 2009.

[50] "General Conditions for a Temporary Arrangement for the Frequency-Dependent Active Power Control of PV Systems in the LV Distribution Network," VDE Association for Electrical, Electronic & Information Technologies, Tech. Rep., 2011.

[51] M. Gottschalk and M. Uslar, "Supporting the Development of Smart Cities using a Use Case Methodology," in *24th International Conference on World Wide Web*. ACM, 2015, pp. 541–545.

[52] B. D. Green, “Integrated Volt VAR Control Centralized,” American Electric Power, Tech. Rep., 2011. [Online]. Available: http://smartgrid.epri.com/UseCases/Integrated%20Volt%20VAR%20Control%20Centralized_ph2add.pdf

[53] C. Greer, D. A. Wollman, D. E. Prochaska, P. A. Boynton, J. A. Mazer, C. T. Nguyen, G. J. FitzPatrick, T. L. Nelson, G. H. Koepke, and A. R. Hefner Jr, “NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 3.0,” *Special Publication (NIST SP)-1108r3*, 2014.

[54] V. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. Hancke, “Smart Grid Technologies: Communication Technologies and Standards,” *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 529–539, 2011.

[55] H. T. Haider, O. H. See, and W. Elmenreich, “A review of residential demand response of smart grid,” *Renewable and Sustainable Energy Reviews*, vol. 59, pp. 166–178, 2016.

[56] A. Hakala-Ranta, O. Rintamaki, and J. Starck, “Utilizing possibilities of IEC 61850 and GOOSE,” in *20th International Conference and Exhibition on Electricity Distribution-Part 1, CIRED.* IET, 2009, pp. 1–4.

[57] I. Hegny, T. Strasser, M. Melik-Merkumians, M. Wenger, and A. Zoitl, “Towards an increased reusability of distributed control applications modeled in IEC 61499,” in *17th IEEE International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, Sep. 2012, pp. 1–8.

[58] G. T. Heineman and W. T. Councill, *Component-Based Software Engineering: Putting the Pieces Together*, 1st ed. Boston, Mass. u.a.: Addison-Wesley Professional, Jun. 2001.

[59] “Helinks LLC,” Nov. 2017. [Online]. Available: http://www.helinks.com/

[60] N. Higgins, V. Vyatkin, N.-K. Nair, and K. Schwarz, “Distributed Power System Automation With IEC 61850, IEC 61499, and Intelligent Control,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 41, no. 1, pp. 81–92, 2011.

[61] Holonic Manufacturing Systems (HMS) Consortium, “IEC 61499 Compliance Profile for Feasibility Demonstrations,” Sep. 2013. [Online]. Available: http://www.holobloc.com/doc/ita/index.htm

[62] J. Hughes, “IntelliGrid Architecture Concepts and IEC61850,” in *2005/2006 IEEE/PES Transmission and Distribution Conference and Exhibition*, May 2006, pp. 401–404.

[63] *IEC 61131-3: Programmable controllers - Part 3: Programming languages.* Geneva, Switzerland: International Electrotechnical Commission (IEC), 2012.

[64] *IEC 61499: Function blocks.* Geneva, Switzerland: Int. Elect. Commission (IEC), 2012.

[65] *IEC 61850-7-420: Communication networks and systems for power utility automation - Part 7-420: Basic communication structure - Distributed energy resources logical nodes.* Geneva, Switzerland: International Electrotechnical Commission (IEC), 2009.

[66] *IEC 61850: Communication networks and systems for power utility automation.* Geneva, Switzerland: International Electrotechnical Commission (IEC), 2010.

[67] *IEC 61968: Application integration at electrical utilities.* Geneva, Switzerland: International Electrotechnical Commission, 2003, no. IEC 61968, published: Part 1 - 14.

[68] *IEC 61970: Energy management system application program interface (EMS-API).* Geneva, Switzerland: International Electrotechnical Commission, 2004, no. IEC 61970, published: Part 1-5.

[69] *IEC 62559, PAS: Intelligrid Methodology for developing requirements for Energy Systems.* Geneva, Switzerland: International Electrotechnical Commission, 2008, no. IEC 62559.

[70] *IEC 62559: Use case methodology.* Geneva, Switzerland: International Electrotechnical Commission, 2015, no. IEC 62559.

[71] *IEC/TR 61850-90-7 - Communication networks and systems for power utility automation - Part 90-7: Object models for power converters in distributed energy resources (DER) systems.* Geneva, Switzerland: International Electrotechnical Commission (IEC), 2013.

[72] "IEEE Guide for Smart Grid Interoperability of Energy Technology and Information Technology Operation with the Electric Power System (EPS), End-Use Applications, and Loads," *IEEE Std 2030-2011*, pp. 1–126, Oct. 2011.

[73] M. D. Ilic, L. Xie, U. A. Khan, and J. M. Moura, "Modeling of future cyber-physical energy systems for distributed sensing and control," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 40, no. 4, pp. 825–838, 2010.

[74] "Interface Definition Language Version 3.5," Object Management Group (OMG), Tech. Rep., 2014. [Online]. Available: http://www.omg.org/spec/IDL35

[75] R. Isermann, J. Schaffnit, and S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," *Control Engineering Practice*, vol. 7, no. 5, pp. 643–653, 1999.

[76] K. H. John and M. Tiegelkamp, *IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids.* Springer Science & Business Media, 2010.

[77] J. Jorgensen, S. Sorensen, K. Behnke, and P. Eriksen, "EcoGrid EU — A prototype for European Smart Grids," in *Power and Energy Society General Meeting, 2011 IEEE*, 2011, pp. 1–7.

[78] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Science of Computer Programming*, vol. 72, no. 1, pp. 31–39, Jun. 2008.

[79] F. Knirsch, D. Engel, C. Neureiter, M. Frincu, and V. Prasanna, "Model-driven privacy assessment in the smart grid," in *International Conference on Information Systems Security and Privacy (ICISSP).* IEEE, 2015, pp. 1–9.

[80] G. Kondrak, "N-Gram Similarity and Distance," in *String Processing and Information Retrieval*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. Consens, and G. Navarro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3772, pp. 115–126, dOI: 10.1007/11575832_13.

[81] F. Kupzog, R. Schwalbe, W. Prüggler, B. Bletterie, S. Kadam, A. Abart, and M. Radauer, "Maximising low voltage grid hosting capacity for PV and electric mobility by distributed voltage control," *e & i Elektrotechnik und Informationstechnik*, vol. Volume 131, Issue 6, pp. 188–192, 2014.

[82] I. Kurtev, "State of the art of QVT: A model transformation language standard," in *Applications of graph transformations with industrial relevance.* Springer, 2008, pp. 377–393.

[83] L. Lednicki and J. Carlson, "A framework for generation of inter-node communication in component-based distributed embedded systems," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sep. 2014, pp. 1–8.

[84] E. A. Lee, "Cyber physical systems: Design challenges," in *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on.* IEEE, 2008, pp. 363–369.

[85] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, 1966, pp. 707–710.

[86] R. W. Lewis, *Modeling control systems using IEC 61499.* IEE Publishing, 2001, no. ISBN: 0 85296 796 9.

[87] "libIEC61850 / lib60870-5 | open source libraries for IEC 61850 and IEC 60870-5-104," 2017. [Online]. Available: http://libiec61850.com/libiec61850/

[88] H. Lin, "Communication infrastructure for the smart grid: A co-simulation based study on techniques to improve the power transmission system functions with efficient data networks," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2012.

[89] M. Liserre, T. Sauter, and J. Hung, "Future Energy Systems: Integrating Renewable Energy Sources into the Smart Power Grid Through Industrial Electronics," *IEEE Industrial Electronics Magazine*, vol. 4, no. 1, pp. 18–37, 2010.

[90] A. J. Lopes, R. Lezama, and R. Pineda, "Model Based Systems Engineering for Smart Grids as Systems of Systems," *Procedia Computer Science*, vol. 6, no. Supplement C, pp. 441–450, Jan. 2011.

[91] J. Lopes, N. Hatziargyriou, J. Mutale, P. Djapic, and N. Jenkins, "Integrating distributed generation into electric power systems: A review of drivers, challenges and opportunities," *Electric Power Systems Research*, vol. 77, no. 9, pp. 1189–1203, 2007.

[92] R. Lopez, A. Moore, and J. Gillerman, "A model-driven approach to Smart Substation automation and integration for Comision Federal de Electricidad," in *IEEE PES T D 2010*, Apr. 2010, pp. 1–8.

[93] R. Mall, *Fundamentals of Software Engineering.* PHI Learning Pvt. Ltd., May 2009.

[94] A. W. McMorran, E. M. Stewart, C. M. Shand, S. E. Rudd, and G. A. Taylor, "Addressing the challenge of data interoperability for off-line analysis of distribution networks in the Smart Grid," in *PES T D 2012*, May 2012, pp. 1–5.

[95] A. W. McMorran, "An introduction to IEC 61970-301 & 61968-11: The common information model," *University of Strathclyde*, vol. 93, p. 124, 2007.

[96] S. J. Mellor, S. Kendall, A. Uhl, and D. Weise, *MDA Distilled.* Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[97] "Meta Object Facility," Object Management Group (OMG), Tech. Rep., 2016. [Online]. Available: http://www.omg.org/spec/MOF/2.5.1/

[98] K. Mets, J. A. Ojea, and C. Develder, "Combining Power and Communication Network Simulation for Cost-Effective Smart Grid Analysis," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1771–1796, 2014.

[99] G. Migliavacca, M. Rossi, D. Six, M. Džamarija, R.-I. E. DTU-Denmark, S. HORSMANHEIMO, C. MADINA, I. Kockar, and J. M. MORALES, "SmartNet: a H2020 project analysing TSO-DSO interaction to enable ancillary services provision from distribution networks," *CIRED, Glasgow*, 2017.

[100] “Model Driven Architecture (MDA): The MDA Guide Rev 2.0,” Object Management Group (OMG), Tech. Rep. ormsc/2014-06-01, Jun. 2014.

[101] “MOF Model to Text Transformation Language,” Object Management Group (OMG), Tech. Rep., 2008. [Online]. Available: http://www.omg.org/spec/MOFM2T/1.0/

[102] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, and J. Hill, “The relevance of model-driven engineering thirty years from now,” in *International Conference on Model Driven Engineering Languages and Systems.* Springer, 2014, pp. 183–200.

[103] C. Neureiter, D. Engel, J. Trefke, R. Santodomingo, S. Rohjans, and M. Uslar, “Towards consistent smart grid architecture tool support: From use cases to visualization,” in *IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe).* IEEE, 2014, pp. 1–6.

[104] “NIST Framework and Roadmap for Smart Grid Interoperability Standards,” National Institute of Standards and Technology - U.S. Department of Commerce, USA, Tech. Rep. NIST Publication 1108, 2010.

[105] “OMG System Modeling Language,” Object Management Group (OMG), Tech. Rep., 2017. [Online]. Available: http://www.omg.org/spec/SysML/1.5/

[106] W. Omona, T. P. van der Weide, and J. T. Lubega, “Knowledge Management Research Using Grounded Theory Strategy: Applicability, Limitations and Ways Forward,” in *ICCIR 10 : Proceedings of the 6th Annual International Conference on Computing and ICT Research*, Jan. 2010, pp. 163–185.

[107] P. Palensky and D. Dietrich, “Demand Side Management: Demand Response, Intelligent Energy Systems, and Smart Loads,” *IEEE Transactions on Industrial Informatics*, vol. 7, no. 3, pp. 381–388, 2011.

[108] T. Parr, *The Definitive ANTLR Reference: Building Domain-specific Languages.* Pragmatic Bookshelf, 2007.

[109] R. Paulo and A. Carvalho, “Towards model-driven design of substation automation systems,” in *Proc. of CIRED*, 2005.

[110] M. Petre, “Why Looking Isn’T Always Seeing: Readership Skills and Graphical Programming,” *Com. ACM*, vol. 38, no. 6, pp. 33–44, 1995.

[111] “Power generation systems connected to the low-voltage distribution network - Technical minimum requirements for the connection to and parallel operation with low-voltage distribution networks,” VDE Association for Electrical, Electronic & Information Technologies, Tech. Rep. VDE-AR-N 4105, 2011.

[112] "PowerFactory - DIgSILENT Germany," 2017. [Online]. Available: http://www.digsilent.de/index.php/products-powerfactory.html

[113] Y. Pradeep, P. Seshuraju, S. A. Khaparde, V. S. Warrier, and S. Cherian, "CIM and IEC 61850 integration issues: Application to power systems," in *2009 IEEE Power Energy Society General Meeting*, Jul. 2009, pp. 1–6.

[114] "Prescriptions techniques spécifiques de raccordement d'installations de production décentralisée fonctionnant en parallèle sur le réseau de distribution," Synergrid, Tech. Rep., 2012.

[115] B. Prindle, M. Eldridge, M. Eckhardt, and A. Frederick, "The twin pillars of sustainable energy: synergies between energy efficiency and renewable energy technology and policy," *Washington, DC: American Council for an Energy-Efficient Economy*, 2007.

[116] F. Pröstl Andrén, T. Strasser, and W. Kastner, "Applying the SGAM Methodology for Rapid Prototyping of Smart Grid Applications," in *42nd Annual Conference of the IEEE Ind. Electronics Society (IECON)*, 2016.

[117] ——, "Engineering Smart Grids: Applying Model-Driven Development from Use Case Design to Deployment," *Energies*, vol. 10, no. 3, p. 374, Mar. 2017.

[118] F. Pröstl Andrén, T. Strasser, O. Langthaler, A. Veichtlbauer, C. Kasberger, and G. Felbauer, "Open and Interoperable ICT Solution for Integrating Distributed Energy Resources into Smart Grids," in *21th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2016)*, 2016.

[119] A. Purkus, E. Gawel, M. Deissenroth, K. Nienhaus, and S. Wassermann, "Market integration of renewable energies through direct marketing - lessons learned from the German market premium scheme," *Energy, Sustainability and Society*, vol. 5, no. 1, p. 12, Apr. 2015.

[120] S. Raman, N. Sivashankar, W. Milam, W. Stuart, and S. Nabi, "Design and implementation of HIL simulators for powertrain control system software development," in *American Control Conference (ACC)*, 1999.

[121] A. Rodrigues da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Computer Languages, Systems & Structures*, vol. 43, no. Supplement C, pp. 139–155, Oct. 2015.

[122] S. Rohjans, C. Dänekas, and M. Uslar, "Requirements for Smart Grid ICT-architectures," in *2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Oct. 2012, pp. 1–8.

[123] S. Rohjans, *Semantic service integration for smart grids.* Ios Press, 2012, vol. 14.

[124] S. Rohjans, S. Lehnhoff, S. Schütte, F. Andrén, and T. Strasser, "Requirements for Smart Grid simulation tools," in *IEEE 23rd International Symposium on Industrial Electronics (ISIE).* IEEE, 2014, pp. 1730–1736.

[125] K. S. Rubin, *Essential Scrum: A Practical Guide to the Most Popular Agile Process.* Addison-Wesley Professional, 2012.

[126] R. Santodomingo, J. A. Rodríguez-Mondéjar, and M. A. Sanz-Bobi, "Ontology matching approach to the harmonization of CIM and IEC 61850 standards," in *First IEEE International Conference on Smart Grid Communications (SmartGridComm).* IEEE, 2010, pp. 55–60.

[127] R. Santodomingo, M. Uslar, A. Göring, M. Gottschalk, L. Nordström, A. Saleem, and M. Chenine, "SGAM-based methodology to analyse Smart Grid solutions in DISCERN European research project," in *2014 IEEE International Energy Conference (ENERGYCON)*, May 2014, pp. 751–758.

[128] D. C. Schmidt, "Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[129] R. Schwalbe, A. Einfalt, A. Abart, M. Radauer, and H. Brunner, "DG DemoNet Smart LV Grid - Robust Control Architecture to Increase DG Hosting Capacity," in *23rd International Conference and Exhibition on Electricity Distribution, CIRED*, 2015.

[130] M. Shaw, "Writing good software engineering research papers," in *25th International Conference on Software Engineering*, May 2003, pp. 726–736.

[131] ——, "What makes good research in software engineering?" *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 4, no. 1, pp. 1–7, 2002.

[132] J. Siegel, "Developing in OMG's New Model-Driven Architecture," *Management*, 2001.

[133] M. Slee, A. Agarwal, and M. Kwiatkowski, "Thrift: Scalable cross-language services implementation," *Facebook White Paper*, vol. 5, no. 8, 2007.

[134] "Smart Grid International Research Facility Network (SIRFN), IEA ISGAN Annex 5," 2017. [Online]. Available: http://www.sirfn.net/

[135] SMB Smart Grid Strategic Group (SG3), "IEC Smart Grid Standardization Roadmap," International Electrotechnical Commission (IEC), Geneva, Switzerland, Tech. Rep. Ed. 1.0, 2010.

[136] H. G. Sol, "Simulation in information systems development." Ph.D. dissertation, University of Groningen, 1982.

[137] J. Somers, "The Coming Software Apocalypse," *The Atlantic*, Sep. 2017. [Online]. Available: https://www.theatlantic.com/technology/archive/2017/09/saving-the-world-from-code/540393/

[138] M. Specht, S. Rohjans, J. Trefke, M. Uslar, and J. M. Gonzalez Vazquez, "International Smart Grid Roadmaps and their Assessment," *EAI Endorsed Transactions on Energy Web*, vol. 1, Mar. 2013.

[139] "Specifications – OSGi™ Alliance," Jan. 2018. [Online]. Available: https://www.osgi.org/developer/specifications/

[140] M. H. Spiegel, F. Leimgruber, E. Widl, and G. Gridling, "On using FMI-based models in IEC 61499 control applications," in *2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, Apr. 2015, pp. 1–6.

[141] T. Stetz, F. Marten, and M. Braun, "Improved Low Voltage Grid-Integration of Photovoltaic Systems in Germany," *IEEE Transactions on Sustainable Energy*, vol. 4, no. 2, pp. 534–542, Apr. 2013.

[142] M. Steurer, F. Bogdan, W. Ren, M. Sloderbeck, and S. Woodruff, "Controller and power hardware-in-loop methods for accelerating renewable energy integration," in *Power Engineering Society General Meeting, 2007. IEEE.* IEEE, 2007, pp. 1–4.

[143] M. Stifter, E. Widl, F. Andrén, A. Elsheikh, T. Strasser, and P. Palensky, "Co-simulation of components, controls and power systems based on open source software," in *2013 IEEE Power Energy Society General Meeting*, Jul. 2013, pp. 1–5.

[144] M. Stifter, F. Andrén, R. Schwalbe, and W. Tremmel, "Interfacing PowerFactory: Co-simulation, Real-Time Simulation and Controller Hardware-in-the-Loop Applications," in *PowerFactory Applications for Power System Analysis*, ser. Power Systems. Springer, Cham, 2014, pp. 343–366, dOI: 10.1007/978-3-319-12958-7_15.

[145] T. Strasser, F. Andrén, J. Kathan, C. Cecati, C. Buccella, P. Siano, P. Leitao, G. Zhabelova, V. Vyatkin, P. Vrba, and V. Marik, "A Review of Architectures and Concepts for Intelligence in Future Electric Energy Systems," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 4, pp. 2424–2438, Apr. 2015.

[146] T. Strasser, F. Andrén, F. Lehfuss, M. Stifter, and P. Palensky, "Online Reconfigurable Control Software for IEDs," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–1, 2013.

[147] T. Strasser, F. Andrén, V. Vyatkin, G. Zhabelova, and C.-W. Yang, "Towards an IEC 61499 compliance profile for smart grids review and analysis of possibilities," in *Annual Conference on IEEE Industrial Electronics Society (IECON)*, 2012, pp. 3750–3757.

[148] T. Strasser, M. Rooker, I. Hegny, M. Wenger, A. Zoitl, L. Ferrarini, A. Dede, and a. M. Colla, “A Research Roadmap for Model-Driven Design of Embedded Systems for Automation Components,” in *IEEE International Conference on Industrial Informatics (INDIN’09)*, 2009.

[149] T. Strasser, M. Stifter, F. Andrén, D. Burnier, and W. Hribernik, “Applying open standards and open source software for smart grid applications: Simulation of distributed intelligent control of power systems,” in *Power and Energy Society General Meeting, IEEE*, 2011.

[150] T. Strasser, M. Stifter, F. Andrén, and P. Palensky, “Co-Simulation Training Platform for Smart Grids,” *IEEE Transactions on Power Systems*, vol. 29, no. 4, pp. 1989–1997, Jul. 2014.

[151] T. Strasser, A. Zoitl, J. Christensen, and C. Sünder, “Design and Execution Issues in IEC 61499 Distributed Automation and Control Systems,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 41, no. 1, pp. 41–51, 2011.

[152] T. Strasser, F. Andrén, R. Bründlinger, and D. Garabandic, “Integrating PV into the Smart Grid - Implementation of an IEC 61850 Interface for Large Scale PV Inverters,” in *EUPVSEC Proceedings.* WIP, 2013.

[153] T. Strasser, F. Pröstl Andrén, G. Lauss, R. Bründlinger, H. Brunner, C. Moyo, C. Seitl, S. Rohjans, S. Lehnhoff, P. Palensky, P. Kotsampopoulos, N. Hatziargyriou, G. Arnold, W. Heckmann, E. Jong, M. Verga, G. Franchioni, L. Martini, A. Kosek, O. Gehrke, H. Bindner, F. Coffele, G. Burt, M. Calin, and E. Rodriguez-Seco, “Towards holistic power distribution system validation and testing—an overview and discussion of different possibilities,” *e & i Elektrotechnik und Informationstechnik*, Dec. 2016.

[154] “Substation Automation Systems - Smart Grid Solutions - Siemens,” Nov. 2017. [Online]. Available: http://w3.usa.siemens.com/smartgrid/us/en/transmission-grid/products/substation-automation-systems/pages/substation-automation-system.aspx

[155] “SystemCORP: Smart Grid Controllers,” Nov. 2017. [Online]. Available: https://www.systemcorp.com.au/products/hardware/smart-grid-controllers/

[156] C. Szyperski, *Component Software: Beyond Object-oriented Programming.* ACM Press, Jan. 1997.

[157] “Technical Guideline: Generating Plants Connected to the Medium-Voltage Network,” BDEW German Association of Energy and Water Industries, Tech. Rep., 2008.

[158] “Technology Roadmap Smart Grids,” International Energy Agency (IEA), Tech. Rep., 2011.

[159] “The German Standardisation Roadmap E-Energy/Smart Grid,” German Commission for Electrical, Electronic & Information Technologies of DIN and VDE, Frankfurt, Germany, Tech. Rep., 2010.

[160] “TOR D4 Parallelbetrieb von Erzeugungsanlagen mit Verteilernetzen,” e-control, Tech. Rep. Version 2.1, 2013.

[161] “Trail: JavaBeans(TM) (The Java™ Tutorials),” Jun. 2017. [Online]. Available: https://docs.oracle.com/javase/tutorial/javabeans/

[162] J. Trefke, J. M. González, and M. Uslar, “Smart Grid standardisation management with use cases,” in *IEEE International Energy Conference and Exhibition (ENERGYCON)*. IEEE, 2012, pp. 903–908.

[163] J. Trefke, S. Rohjans, M. Uslar, S. Lehnhoff, L. Nordström, and A. Saleem, “Smart Grid Architecture Model use case management in a large European Smart Grid project,” in *Innovative Smart Grid Technologies Europe (ISGT EUROPE), 2013 4th IEEE/PES*. IEEE, 2013, pp. 1–5.

[164] “Unified Modeling Language,” Object Management Group (OMG), Tech. Rep., 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5/

[165] M. Uslar, F. Andrén, W. Mahnke, S. Rohjans, M. Stifter, and T. Strasser, “Hybrid grids: ICT-based integration of electric power and gas grids-A standards perspective,” in *3rd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies (ISGT Europe)*. IEEE, 2012, pp. 1–8.

[166] M. Uslar, M. Specht, S. Rohjans, J. Trefke, and J. M. González, *The Common Information Model CIM: IEC 61968/61970 and 62325-A practical introduction to the CIM*. Springer Science & Business Media, 2012.

[167] A. Viehweider, G. Lauss, and F. Lehfuss, “Stabilization of Power Hardware-in-the-Loop simulations of electric energy systems,” *Simulation Modelling Practice and Theory*, vol. 19(2011), pp. 1699–1708, 2011.

[168] V. Vyatkin, “The IEC 61499 standard and its semantics,” *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 40–48, 2009.

[169] ——, “IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review,” *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 768–781, 2011.

[170] V. Vyatkin, G. Zhabelova, N. Higgins, K. Schwarz, and N. Nair, “Towards intelligent Smart Grid devices with IEC 61850 Interoperability and IEC 61499 open control architecture,” in *2010 IEEE PESTransmission and Distribution Conference and Exposition*, 2010.

[171] V. Vyatkin, G. Zhabelova, N. Higgins, M. Ulieru, K. Schwarz, and N. Nair, “Standards-enabled Smart Grid for the future Energy Web,” in *Innovative Smart Grid Technologies (ISGT), 2010*, 2010, pp. 1–9.

[172] D. Westermann and M. Kratz, “A real-time development platform for the next generation of power system control functions,” *IEEE Transactions on Industrial Electronics*, vol. 57, no. 4, pp. 1159–1166, 2010.

[173] “World Energy Outlook 2013,” Intern. Energy Agency, Tech. Rep., 2013.

[174] C.-W. Yang, V. Vyatkin, A. Mousavi, and V. Dubinin, “On automatic generation of IEC61850/IEC61499 substation automation systems enabled by ontology,” in *40th Annual Conf. of the IEEE Ind. Electronics Society (IECON)*, 2014.

[175] L. Zhu, D. Shi, and X. Duan, “Standard Function Blocks for Flexible IED in IEC 61850-Based Substation Automation,” *IEEE Transactions on Power Delivery*, vol. 26, no. 2, pp. 1101–1110, 2011.

[176] A. Zoitl, T. Strasser, and A. Valentini, “Open source initiatives as basis for the establishment of new technologies in industrial automation: 4diac a case study,” in *2010 IEEE International Symposium on Industrial Electronics (ISIE)*, 2010.

[177] A. Zoitl and V. Vyatkin, “IEC 61499 architecture for distributed automation: The “glass half full” view,” *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 7–23, 2009.

[178] A. Zoitl and H. Prahofer, “Guidelines and Patterns for Building Hierarchical Automation Solutions in the IEC 61499 Modeling Language,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2387–2396, Nov. 2013.

Filip Pröstl Andrén — Curriculum Vitae

CONTACT INFORMATION

Filip Pröstl Andrén

Oskar-Jascha-Gasse 61
1130 Wien
Austria
M +43 664 2351916
filip.proestl-andren@ait.ac.at

PERSONAL DATA

Full name	Per Johan Filip Pröstl Andrén
Nationality	Swedish
Born	March 13th, 1984 in Helsingborg, Sweden
Languages	Swedish (native), German (fluid), English (fluid)

EDUCATION

Vienna University of Technology, Vienna, Austria

PhD, Informatics **since June 2013**

- Thesis *Model-Driven Engineering for Smart Grid Automation*
  - ∗ Supervisor: Wolfgang Kastner

Linköping University, Linköping, Sweden

MSc, Applied Physics and Electrical Engineering **Dec. 2009**

- Thesis *Optimization of Random Access in 3G Long Term Evolution*
  - ∗ Supervisor: Fredrik Gunnarsson

Fachhochschule Technikum Wien, Vienna, Austria

Exchange Student, Mechatronics/Robotics **Sept. 2008 – Feb. 2009**

EMPLOYMENT RECORD / ACADEMIC EXPERIENCE

AIT Austrian Institute of Technology, Center for Energy, Electrical Energy Systems, Vienna, Austria

Scientist **since Sept. 2014**

- Topics of research: control and information systems for smart grids, power utility automation, model-driven engineering

University of Applied Sciences Technikum, Department of Renewable Energy, Vienna, Austria

Lecturer **since Mar. 2012**

- Tutorial: Anlagentechnik und Simulation - Smart Grids

AIT Austrian Institute of Technology, Center for Energy, Electric Energy Systems, Vienna, Austria

Junior Scientist **Nov. 2009 – Sept. 2014**

- Topics of research: control and information systems for smart grids, power utility automation, power systems modeling and simulation

Publications Most Recent and Relevant

[1] C. Zanabria, A. Tayyebi, F. Pröstl Andrén, J. Kathan, and T. Strasser, "Engineering support for handling controller conflicts in energy storage systems applications," *Energies*, vol. 10, no. 10, 2017.

[2] F. Pröstl Andrén, T. Strasser, and W. Kastner, "Engineering Smart Grids: Applying Model-Driven Development from Use Case Design to Deployment," *Energies*, vol. 10, no. 3, p. 374, Mar. 2017.

[3] M. Faschang, S. Cejka, M. Stefan, A. Frischenschlager, A. Einfalt, K. Diwold, F. Pröstl Andrén, T. Strasser, and F. Kupzog, "Provisioning, deployment, and operation of smart grid applications on substation level," *Computer Science - Research and Development*, vol. 32, no. 1, pp. 117–130, Mar 2017.

[4] T. Strasser, F. Pröstl Andrén, G. Lauss, R. Brndlinger, H. Brunner, C. Moyo, C. Seitl, S. Rohjans, S. Lehnhoff, P. Palensky, P. Kotsampopoulos, N. Hatziargyriou, G. Arnold, W. Heckmann, E. Jong, M. Verga, G. Franchioni, L. Martini, A. Kosek, O. Gehrke, H. Bindner, F. Coffele, G. Burt, M. Calin, and E. Rodriguez-Seco, "Towards Holistic Power Distribution System Validation and Testing—An Overview and Discussion of Different Possibilities," *e & i Elektrotechnik und Informationstechnik*, Dec. 2016.

[5] F. Pröstl Andrén, T. Strasser, and W. Kastner, "Applying the SGAM Methodology for Rapid Prototyping of Smart Grid Applications," in *42nd Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2016.

[6] F. Pröstl Andrén, T. Strasser, O. Langthaler, A. Veichtlbauer, C. Kasberger, and G. Felbauer, "Open and Interoperable ICT Solution for Integrating Distributed Energy Resources into Smart Grids," in *21th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2016)*, 2016.

[7] F. Andrén, B. Bletterie, S. Kadam, P. Kotsampopoulos, and C. Bucher, "On the Stability of Local Voltage Control in Distribution Networks With a High Penetration of Inverter-Based Generation," *IEEE Transactions on Industrial Electronics*, vol. Volume 62, Issue 4, pp. 2519–2529, 2015.

[8] T. Strasser, F. Andrén, J. Kathan, C. Cecati, C. Buccella, P. Siano, P. Leitao, G. Zhabelova, V. Vyatkin, P. Vrba, and V. Marik, "A Review of Architectures and Concepts for Intelligence in Future Electric Energy Systems," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 4, pp. 2424–2438, Apr. 2015.

[9] F. Andrén, T. Strasser, and W. Kastner, "From Textual Programming to IEC 61499 Artifacts - Towards a Model-Driven Engineering Approach for Smart Grid Applications," in *13th IEEE International Conference on Industrial Informatics (INDIN 2015)*, IEEE, Ed. IEEE, 2015, pp. 1524–1530.