UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

PAULO NEIS

DOMAIN-SPECIFIC POWER APPLICATIONS DEVELOPMENT ENVIRONMENT AND STRATEGY: A MODEL-DRIVEN APPROACH TO SCADA/EMS

TESE

CURITIBA
2023

PAULO NEIS

DOMAIN-SPECIFIC POWER APPLICATIONS DEVELOPMENT ENVIRONMENT AND STRATEGY: A MODEL-DRIVEN APPROACH TO SCADA/EMS

Ambiente e Estratégias Específicas do Domínio para o Desenvolvimento de Aplicações de Potência: Uma Abordagem Guiada por Modelos

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná (UTFPR), como requisito parcial para obtenção do título de "Doutor em Ciências" - Área de Concentração: Engenharia de Computação.

Orientador: Prof. Dr. Marco Aurélio Wehrmeister

CURITIBA

2023

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Curitiba

PAULO NEIS

DOMAIN-SPECIFIC POWER APPLICATIONS DEVELOPMENT ENVIRONMENT AND STRATEGY: A MODEL-DRIVEN APPROACH TO SCADA/EMS.

Trabalho de pesquisa de doutorado apresentado como requisito para obtenção do título de Doutor Em Ciências da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Computação.

Data de aprovação: 05 de Maio de 2023

Dr. Marco Aurelio Wehrmeister, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Carlos Eduardo Pereira, Doutorado - Universidade Federal do Rio Grande do Sul (Ufrgs)

Dr. Paulo Cezar Stadzisz, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Rodrigo Andrade Ramos, Doutorado - Usp-Universidade de São Paulo

Dr. Rui Jovita Godinho Correa Da Silva, Doutorado - Fundação Parque Tecnológico Itaipu

RESUMO

NEIS, Paulo. **Ambiente e Estratégias Específicas do Domínio para o Desenvolvimento de Aplicações de Potência: Uma Abordagem Guiada por Modelos**. 2023. 178 f. Tese (Doutorado em Engenharia Elétrica e Informática Industrial) – Universidade Tecnológica Federal do Paraná. Curitiba, 2023.

Esta tese propõe uma abordagem de Engenharia Guiada por Modelos para o desenvolvimento de aplicações que compõem uma suíte conhecida como Sistema de Gerenciamento de Energia (EMS). EMS é uma ferramenta essencial para a operação dos sistemas elétricos de potência, incluindo usinas hidrelétricas. Devido às características e complexidade de cada projeto, produtos comerciais "de prateleira" podem não ser capazes de atender a todos os requisitos, demandando o desenvolvimento de customizações, ou mesmo aplicações inteiramente novas para cada fornecimento. A gestão do ciclo de vida de softwares customizados torna-se complexa, particularmente durante atualizações, migrações para novo produto, ou troca de fornecedor. O software EMS faz parte de um sistema ciber-físico, para o qual sofisticados modelos costumam ser construídos durante as fases de planejamento, projeto e operação das instalações. Tais modelos, porém, costumam ser utilizados no processo de desenvolvimento apenas como suporte e documentação, sendo que o código-fonte é considerado o principal artefato do software EMS. A evolução de tais sistemas torna-se dispendiosa e sujeita a erros, requerendo modificações manuais no código-fonte e documentação correspondente. A Engenharia Guiada por Modelos (MDE) é uma abordagem para o projeto de sistemas na qual os modelos são utilizados como artefatos principais no processo de desenvolvimento, enquanto os programas (e outros produtos) são automaticamente gerados a partir destes. Abordagens MDE têm sido adotadas em diversas áreas da indústria, para as quais existem linguagens e ferramentas especializadas que facilitam as atividades de modelagem e transformação automática dos modelos em software executável. Porém, tratando-se de desenvolvimento de software SCADA/EMS, ainda inexiste uma abordagem dedicada. Esta tese propõe o D-SPADES: Ambiente e Estratégias Específicas do Domínio para o Desenvolvimento de Aplicações de Potência, uma abordagem de Engenharia Guiada por Modelos feita sob medida para o domínio de aplicações EMS. D-SPADES dispõe de uma linguagem de modelagem específica para este domínio, baseada em uma notação de diagramas de bloco, associada com estratégias de mapeamento e ferramentas para transformar automaticamente modelos em código fonte de aplicações que podem ser integradas em plataformas SCADA existentes. D-SPADES foi aplicado para modelar e gerar automaticamente o código-fonte de duas aplicações reais na Usina de Itaipu: um controlador automático de volt/var, e um componente para um sistema especial de proteção. O novo controlador de volt/var foi validado criteriosamente utilizando-se um simulador de sistemas de potência, e comparado à uma aplicação existente, que foi desenvolvida utilizando abordagens tradicionais. O desempenho do novo controlador em termos de funcionalidades, uso de recursos e métricas de código fonte foi considerado satisfatório. O sistema especial de proteção, após ser submetido à uma série de testes sistêmicos, tanto reais quanto simulados, foi implantado definitivamente em ambiente de produção, e encontra-se plenamente operacional desde 2022. Desta forma, foi demonstrado que D-SPADES é uma abordagem viável para o desenvolvimento de aplicações SCADA/EMS de missão crítica.

Palavras-chave: Engenharia de Software. Engenharia Guiada por Modelos. usinas hidrelétricas. sistemas elétricos de potência. sistemas de controle supervisório.

ABSTRACT

NEIS, Paulo. **Domain-Specific Power Applications Development Environment and Strategy: A Model-Driven Approach to SCADA/EMS**. 2023. 178 p. Thesis (PhD in Graduate Program in Electrical and Computer Engineering) – Universidade Tecnológica Federal do Paraná. Curitiba, 2023.

This thesis proposes a model-driven approach to develop applications that compose a suite known as Energy Management System (EMS). EMS is an essential tool for the operation of electrical systems, including hydroelectric power plants. Given the characteristics and complexity of each project, commercial "off-the-shelf" products might not fulfill all the requirements, demanding the development of customized versions or even entirely new applications for each customer. Managing the life cycle of such customized software becomes a complicated endeavor, particularly while performing upgrades, or migrating to a different vendor or product line. EMS software constitutes part of a cyber-physical system, for which sophisticated models are often constructed during the planning, designing, and operation of the installations. Such models, however, are usually employed only as support and documentation, while source code ends up being the main EMS software artifact throughout the development process. The evolution of such a system is expensive and error-prone, requiring manual changes to source code and the corresponding documentation. Model-Driven Engineering (MDE) is an approach to system engineering in which models are used as primary artifacts, while programs (and other products) are automatically generated from such models. MDE approaches are applied in several industrial areas, for which specialized languages and tools support the modeling activities and the automatic transformation of such models into executable software. For SCADA/EMS software development, however, a dedicated approach is still absent. Therefore this thesis proposes D-SPADES: the Domain-Specific Power Applications Development Environment and Strategies, a Model-Driven Engineering approach tailored to the EMS domain. D-SPADES relies on a domain-specific modeling language based on block diagram notation, associated with mapping strategies and tools for automatically transforming models into source code for applications that can be integrated into existing SCADA platforms. D-SPADES have been applied to model and automatically generate source code for two real-world applications at the Itaipu Power Plant: a volt/var controller and a component for a system-wide special protection scheme. The new volt/var controller was thoroughly validated using a power system simulator, and compared to a legacy application, developed through traditional approaches. Its performance in terms of functionality, resource usage, and source code metrics is considered satisfactory. The special protection scheme, after successfully passing through a set of simulated and real system tests, was permanently deployed to the production system and is fully operational since 2022. Hence it has been demonstrated that D-SPADES is a viable approach to the development of mission-critical SCADA/EMS applications.

Keywords: Software Engineering. Model Driven Engineering. hydroelectric power plants. power systems. supervisory control systems.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

INITIALISM	
3GL	Third-generation Programming Language
ABI	Application Binary Interface
AGC	Automatic Generation Control
ALFC	Automatic Load Frequency Controller
AMoDE-RT	Aspect-oriented Model-Driven Engineering for Real-Time Systems
AO	Actor-Oriented Programming Model
AOSD	Aspect-Oriented Software Development
API	Application Program Interface
AQL	Acceleo Query Language
AVC	Automatic Voltage Control
AVR	Automatic Voltage Regulator
CCR	Central Control Room
CF	Constant Frequency control mode
CGH	Central Geradora Hidrelétrica
CIM	Common Information Model
CNI	Constant Net Interchange
COTS	Commercial Off-The-Shelf
CPS	Cyber-Physical Systems
CT	Continuous-Time Model of Computation
D-SPADES	Domain-Specific Power Applications Development Environment and Strategy
DA	Data Analytics
DC	Direct Current
DG	Distributed Generator
DSL	Domain Specific Language
DSML	Domain Specific Modeling Language
DTS	Dispatcher Training Simulator
EMF	Eclipse Modeling Framework
EMS	Energy Management System
EMSML	Energy Management Systems Modeling Language
EMSOnto	Energy Management System Ontology
ERG60	Esquema de Redução de Geração em 60 Hz
FSM	Finite State Machine
GIS	Gas Insulated Switchgear

HVDC	High Voltage Direct Current
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
IED	Intelligent Electronic Devices
IIR	Infinite Impulse Response
IoT	Internet of Things
JBVRC	Joint Bus Voltage/Reactive Control
JCAP	Joint Control of Active Power
JTC	Joint Turbine Control
JVC	Joint Voltage Control
LCR	Local Control Room
LDC	Load Dispatch Center
LFC	Load Frequency Control
M2M	Model-to-Model
M2T	Model-to-Text
MARTE	Modeling and Analysis of Real-Time and Embedded systems
MAS	Multi Agent System
MDE	Model Driven Engineering
MDERG	Model-Driven ERG60
ML	Machine Learning
MMS	Manufacturing Message Specification
MoC	Model of Computation
MOFM2T	MOF Model to Text Transformation Language
MoML	Modeling Markup Language
MPC	Model Predictive Control
ODE	Ordinary Differential Equation
OMG	Object Management Group
OO	Object-Oriented
OPC-UA	OPC Unified Architecture
OS	Operating System
PCH	Pequena Central Hidrelétrica
PLC	Programmable Logic Controller
PN	Process Network Model of Computation
PSAL	Power System Automation Language
PSS	Power System Stabilizer
RAS	Remedial Action Scheme
RPC	Remote Procedure Call
RTU	Remote Terminal Unit
SCADA	Supervisory, Control and Data Acquisition System

SCADA/EMS	Supervisory Control and Data Acquisition / Energy Management System
SDF	Synchronous Dataflow Model of Computation
SEP-765	Sistema Especial de Proteção 765 kV
SGAM	Smart Grid Architecture Model
SLOC	Source Lines Of Code
SysML	Systems Modeling Language
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language

CONTENTS

1 INTRODUCTION

Reliable power systems operations nowadays depend on computerized digital control systems known as *Supervisory Control and Data Acquisition / Energy Management System (SCADA/EMS)*. The so-called *Energy Management System (EMS) applications*[1] perform tasks such as *short-term hydro scheduling*, *joint voltage and reactive power control*, and *automatic generation control*. They are typically executed at the power system's control centers and also in large power plants' control rooms (OLIVEIRA *et al.*, 2017; CASTRO *et al.*, 1992; JALEELI *et al.*, 1992).

Large hydropower plants are the primary source of electricity for the Brazilian grid. According to official reports from the Brazilian National Electricity Agency, more than 50% of the installed generating capacity corresponds to this type of source (ANEEL - Agência Nacional de Energia Elétrica, 2022). Table 1 gives detailed information on the current electricity mix, by type of source, in the Brazilian grid. Notice that plants designated as *Pequena Central Hidrelétrica (PCH)* (rated between 5 and 30 MW), and *Central Geradora Hidrelétrica (CGH)* (rated below 5 MW) correspond to less than 3.5% of the total installed capacity, while large hydropower plants correspond to roughly 55% - thus our emphasis on the importance of this type of source for the Brazilian grid. A relevant consideration is that, according to the ANEEL information system, only the 60Hz portion of Itaipu Power Plant is accounted for in this electricity mix, which corresponds to 7 GW - half of its installed capacity. The energy supplied by the 50Hz sector is accounted as import, therefore this portion (7 GW) is not even considered for the Brazilian electricity mix. In other words, the overall importance of large hydropower plants for the Brazilian grid is even greater than the proportion conveyed in Table 1.

Therefore it can be said that the bulk of the electricity delivered to consumers in Brazil originates from large hydropower stations, some of which are rated among the biggest currently operating in the world. Some of these plants are operational for more than three decades, having been subject to different levels of overhauls. Overhauls and modernizations are usually intended to improve reliability, increase production, reduce operational costs, and also comply with

[1] In this work we employ the term "EMS" primarily to refer to software components performing control and support functions at the centralized and off-site level of power plant control, according to the hierarchical structure classified in the IEEE 1249-2013 Standard (IEC/IEEE, 2013), discussed in Section 3.3. A more general definition of EMS, as discussed in Section 3.3.3, might include several other system-wide grid analysis functions, like state estimation and contingency analysis.

Table 1 – Brazilian electricity mix by the end of year 2022.

Type	Number	Capacity (kW)	Percentage
Large Hydro	**215**	**103,195,357.00**	**54.63%**
Thermo	3141	46,548,804.41	24.64%
Wind	880	23,577,523.86	12.48%
Photovoltaic	17904	7,078,613.67	3.75%
Hydro (PCH)	428	5,662,018.57	3.00%
Nuclear	2	1,990,000.00	1.05%
Hydro (CGH)	719	861,390.42	0.46%
Total	23289	188,913,707.93	100%

Source: ANEEL - Agência Nacional de Energia Elétrica (2022).

regulatory requirements (MENDES, 2011).

Although the technology behind the process of transforming hydraulic energy into electricity and delivering it to the customers (e.g., turbines, generators, switchgear, transformers, and power lines - designated as the *primary system* (MENDES, 2011)) remained essentially unchanged during these decades, the process control and protection (designated as the *secondary system*) has changed radically. The current digital technology applied to control and protection includes programmable controllers, intelligent protective devices, data acquisition, and monitoring equipment. Those components are all interconnected through a complex data network, usually orchestrated by a computerized central control system: the SCADA/EMS. The digital secondary system fits the description of Cyber-Physical Systems (CPS) (LEE; SESHIA, 2011) in the sense that it integrates computation with physical processes through feedback control loops where the processes affect computations and vice versa.

The lifespan of digital control systems is usually shorter than that of electromechanical equipment (MENDES, 2011), for several reasons. In other words, secondary systems change more frequently than primary systems. Therefore, during the power plant's life span, its control system is expected to go through several upgrades. A similar rationale applies to the power system as a whole, including transmission and distribution infrastructure. In the power system's community it is generally accepted that the expected lifespan of a SCADA/EMS system is in the range of 5–10 years (KUIJLAARS, 2015). In our experience at the Itaipu power plant, upgrades have been performed approximately every 10 years. In this scenario of frequent upgrades, control principles are not necessarily changed, but the target platform (controllers, computer systems) may change radically. Control functions developed for a given platform may be incompatible with the upgraded system, possibly having to be re-implemented from scratch. Re-implementing functions, besides increasing cost and development time, also presents risks of introducing new software defects.

Ideally, it should be possible to reuse, in future control system upgrades, functions that were previously well-specified and verified. Another desirable feature is the possibility of automatically transforming the domain-specific models representing the power plant control systems into software artifacts for a given target platform. A complementary feature of this scenario is the possibility of performing improvements to the control functions using a high-level, domain-specific language, independent of the underlying platform and computer programming models.

Model Driven Engineering (MDE) is an approach to system engineering that can help tackle challenges such as those mentioned above. The following two concepts are central to MDE (SCHMIDT, 2006):

1. **Domain-specific modeling languages** that can formally represent the application's requirements, structure, behavior, and possibly other technical aspects within a particular domain, such as the power plant control system;

2. **Transformation engines and generators** that can process models described using the above languages, and produce various types of software artifacts, such as source code, simulation inputs, Extensible Markup Language (XML) deployment descriptions, or alternative model representations. Software artifacts produced by automatic model transformation ensure consistency between the design and actual implementations.

MDE consists in constructing high-level abstract models for the system under development, and then systematically transform them into concrete implementations (FRANCE; RUMPE, 2007; SELIC, 2003). The idea of "concrete implementations" may include VHDL code (LEITE; WEHRMEISTER, 2016), program source code (WEHRMEISTER *et al.*, 2013) (and consequently executable programs), configuration files (WEHRMEISTER *et al.*, 2014), or several other types of artifacts. By employing an abstract domain-specific representation and automatically transforming it into lower-level artifacts, MDE can help cope with the complexity involved in developing software for one particular target platform. Moreover, if the target platform changes, the model remains valid and the software can be more easily ported to the new platform. This portability, however, often depends on the use of standardized middleware, Application Program Interfaces (APIs) or frameworks, rather than lower-level Operating System (OS) APIs.

1.1 SUBJECT OF STUDY

Given the characteristics and complexity of each hydroelectric project, some of the EMS functions need to be customized or developed in a "tailor-made" fashion for that particular project. In these cases, the deployed systems are not the standard Commercial Off-The-Shelf (COTS) products supplied by vendors in the energy management market and may include modifications to the original software, new modules developed for the project, integrations with legacy systems, and other third-party software packages. These customized solutions, although sometimes inevitable, bring serious disadvantages for the product's life cycle management (VAN-SLYKE, 2015), such as:

- Higher cost for initial purchase;
- Higher maintenance contract costs;
- Risk of introducing software defects due to customization;
- Future system upgrades are challenging.

Particularly in the event of a system upgrade, or possibly a migration to a different product line, the impact of customized solutions becomes more dramatic. Modifications made to specific modules on a given release may not be applicable to the newer release and may need a complete redesign. Extra modules integrated into one release of the product may be difficult to port to a different release, or even impossible to migrate into a different product line.

One particular example, Itaipu's digital control system (SCADA/EMS), was deployed in the early 2000s and has already undergone two upgrades. It was supplied by a well-known vendor of energy management and automation systems and includes a considerable degree of customization and special functions developed exclusively for the project. A certain degree of customization was deemed necessary given the particular control requirements dictated by the characteristics and complexity of the Itaipu project. A significant portion of the customized software originally deployed in early 2000, including voltage and generation control, has survived these upgrades, although these applications have evolved, with new functionalities being included along this period. Since the original deployment, these applications have been executed on top of three different SCADA, consisting of different hardware, operating system, and software version, and a fourth one is already on the roadmap.

Therefore, this study concerns methods and technologies for the modeling, development, and maintenance of EMS applications, as well as its integration with the centralized digital control, or *Supervisory, Control and Data Acquisition System (SCADA)* layer. More specifically, we evaluate the applicability of **Model Driven Engineering** approaches to the development and maintenance of these applications. Such approaches can cope with the complexity, mitigate risks and reduce costs of the life cycle management of the SCADA/EMS. This work focuses on large hydropower plant applications and extensively refers to practical examples drawn from the Itaipu Power Station. However, the approach proposed here can be extended and applicable to other power system installations, for instance: power plants based on other energy sources (e.g., thermal, wind, and solar farms), large substations, regional off-site control centers, and systems operators' dispatch centers.

1.1.1 Research Questions

The subject of this research relates to both the software engineering and supervisory control / energy management disciplines. By developing this project, we have addressed concerns that pervade these two areas:

From the Energy Management perspective, particularly concerning the development and evolution of EMS applications, we have formulated answers to the following research questions:

1. Can we choose an appropriate format for modeling EMS applications? The meaning of "appropriate" in this context is that the models should express domain-specific concepts, be reusable between system upgrades, easily extensible and modifiable by the domain specialists.

2. How can the proposed approach contribute to improve model verification and validation (V&V) (reduce "bugs" - design or implementation defects)?

3. Is it possible to automate the transformation of these models into executable artifacts? (the transformation should be at least partially automated, and should involve a minimum effort in writing program code).

4. Is it possible to integrate these executable artifacts with both the existing and future generations of the centralized control systems? (possibly by means of standard protocols

or APIs).

From the Software Engineering perspective, we have investigated and applied MDE techniques to energy management software development, particularly seeking answers to the following questions:

1. Is it possible to model the behavior and structure of EMS applications using a high-level domain-specific language? What kind of language is indicated for this application domain? The languages must offer the resources for correctly and unambiguously specifying the desired behavior, and at the same time abstracting low-level platform details from the modeler.

2. Is it possible to reuse existing/legacy artifacts (specifications, block diagrams, or programs written in third-generation languages) in the model construction, or should the model be rewritten from scratch? We expect that documented block diagrams and transfer functions of the current systems design can be expressed in the proposed language with few changes. Existing code implementing self-contained operations can be also reused.

3. Which transformation techniques and tools can be used to translate the requirements and specified behavior from a high-level modeling language into software artifacts?

4. How can these software artifacts be integrated with existing commercial supervisory control systems? This involves integration with industry standards specific to power systems, particularly power plant control.

1.2 THESIS GOALS

The goal of this research is to propose a novel approach to the development of EMS software, based on MDE concepts. This approach provides a higher level of abstraction for the construction of EMS applications, above the programming language layer, offering advantages regarding both productivity and software quality. It enables the generation of executable software artifacts from high-level models of power system processes. The approach is supported by a set of tools that can be used systematically in the development of this category of applications. The approach can be used either by utilities, software providers, and integrators in the power system's area. It can also be further extended to other categories of industrial applications, such

as automation software, PLC programming, simulation packages, or other technical fields where the same type of abstract models are applicable.

1.2.1 Specific Deliverables

The following deliverables are produced with the development of this project:

1. A basic software process applicable to EMS software development is delineated, identifying the main activities involved.

2. A domain-specific language for modeling energy management applications is proposed, with both abstract and concrete syntaxes.

3. A set of software tools that can support the development process is identified, including the modeling environment, simulation, transformation engines, compilers, and libraries. Additional artifacts needed for a complete and functional development environment, like transformation templates and software component libraries, are also developed.

4. The applicability of the proposed methodology and development environment is demonstrated through both proof-of-concept and real-world applications from the Itaipu power plant. These applications have shown satisfactory logical and timing performance.

1.3 CONTRIBUTIONS AND RELEVANCE

In this research, we propose an MDE-based approach called **Domain-Specific Power Applications Development Environment and Strategy (D-SPADES)** consisting of process, languages, and tools that are tailored to develop and maintain automatic, centralized, **EMS** applications intended to run on top of the SCADA of hydropower plants. D-SPADES can be viewed as a platform, according to the definition given by Lee *et al.* (2002), in the sense that it provides additional layers of abstraction in the design flow of power plant supervisory and control applications. D-SPADES is established upon the following components:

1. An actor-oriented Domain Specific Modeling Language (DSML) called **Energy Management Systems Modeling Language (EMSML)** tailored for modeling EMS applications. This language is designed to be “intuitive” in the sense that it is composed of elements and concepts familiar to the power plant specialists, which are not necessarily literate in

software development techniques. EMSML allows power system specialists to express their intended EMS application design, without necessarily requiring computer or software engineering skills. The models specified through EMSML follow the actor-oriented paradigm. They conform to the EMSML metamodel definition, which consists of a simplified version of the MoML metamodel (LEE; NEUENDORFFER, 2000). In practice, EMSML uses a specialized and more restricted set of the "*Ptolemy II*" (LEE, 2014) actors and models of computation.

2. The **mapping strategies for model transformation** from actor-oriented to object-oriented and/or procedural programming models, which are applied in order to generate source code and produce functional executable artifacts.

3. The **tool support** for the approach, including modeling tools, transformation languages, actors implementation, and SCADA integration libraries. We have demonstrated the applicability of D-SPADES using a chain of tools including: '*Ptolemy II*" modeling and simulation environment, Eclipse Modeling Framework (EMF) [2] tools like the ATL and MTL languages, along with the Acceleo[3] tool for source code generation. Finally, the generated source code is processed by the compiler toolchain of the target language/platform to produce the deployable artifacts. Integration with the underlying SCADA can be achieved using either the proprietary vendor's API or the open standard IEC-625141/OPC-UA.

4. A **software process**, consisting of a coherent set of activities for EMS software production. Such activities include the specification, design, and implementation flow (including integration techniques with existing systems), validation, and software evolution. The successful application of D-SPADES involves the coordinated execution of all these activities.

1.3.1 Originality

We have performed a systematic literature review of published works related to the design and development of EMS functions and other closely related applications (NEIS *et al.*, 2019). This review is extended and further discussed in Chapter 4. In that survey, we have collected evidence that such applications are more commonly modeled using concepts specific

[2] https://www.eclipse.org/modeling/emf/
[3] https://www.eclipse.org/acceleo/

to the power systems' domain, like block diagram notations and transfer functions, rather than traditional techniques and tools from the software industry, like the Unified Modeling Language (UML). In other words, the domain-specific concepts used in this area are different from traditional software modeling notations. The transformation of such models into software implementations, however, is a subject that still needs further research. We have also verified that few studies have investigated MDE approaches to the development of EMS applications, as will be further discussed in Chapter 4.

The D-SPADES approach thus proposes to fill these gaps by:

- Proposing a modeling language and components library enriched with domain-specific concepts, so that power system specialists may more easily participate in the EMS applications' development process;
- Providing an approach for automatically transforming these models into software implementations.

We were unable to identify any previous publications specifically dealing with these problems, and with the level of detail described in this thesis.

1.3.2 Relevance

Some of the biggest hydropower plants in Brazil are operational for more than three decades. Overhauls and upgrades to these plants' control systems are eventually necessary in order to improve reliability, increase production while reducing operational costs, and also comply with regulatory requirements (MENDES, 2011). As explained above, a great deal of the Brazilian electricity demand is met by hydropower plants. Therefore reducing the development cost and improving the reliability of the plant's operations are relevant contributions to the research community and general society.

Specific motivations for this research arise from immediate needs and concerns observed during the ongoing and planned secondary systems upgrades in the Itaipu power plant[4], although other installations in the power sector or even different industrial activities may benefit from such an approach. From our experience in the Itaipu operation, we have observed that COTS SCADA/EMS products are unable to satisfactorily conform to operational requirements, thus

[4] Information about the Itaipu modernization plan is available at: http://www.itaipu.gov.br/en/technology/technological-upgrade.

demanding customization and shared development among the provider and the customer. Such specialized and customer-unique applications on the other hand may potentially offer strategic advantages to the organization.

D-SPADES has already proved useful in the development activities related to the upgrade of a system-wide special protective scheme in the Itaipu power plant, as will be further discussed in Chapter 6. With the ongoing modernization of the Itaipu power plant, which includes the deployment of a brand new SCADA/EMS, we expect that other projects may be conducted using D-SPADES.

1.4 SCOPE DELIMITATION

This work proposes a Model Driven Engineering approach to the development of EMS applications for hydro power plants. We investigate matters related to:

- High-level domain-specific languages that can be used to model power plant control functions, as well as other power systems applications.
- Transformation engines and generators capable of processing high-level models and producing software artifacts. The term “high-level models” in this work refers to “levels of abstraction above the code level” (FRANCE; RUMPE, 2007), particularly those that are based on paradigms used at problem-level modeling.
- The necessary infrastructure for integrating and deploying the above-mentioned software artifacts along with existing centralized digital control systems, including standardized API, middlewares, and/or communication protocols.
- A software process suited for this kind of application, particularly with the capability of accommodating future evolutions of both the control functions and the third party’s centralized control system.

This work is **not** primarily intended to:

- Contribute with control theory or elaborate novel controller models;
- Compare or evaluate control strategies and controller performance;
- Design, evaluate or compare commercial SCADA software or hardware platforms.

Additionally, we emphasize that most of the developments presented in this thesis do not yet constitute finished or commercial products. Although we present one case study where the application of D-SPADES resulted in production-grade software, this is not the main objective of this research.

1.5 ORGANIZATION

The remainder of this thesis is organized as follows: Chapter 2 presents some of the fundamental notions of models and modeling in science and engineering. This chapter helps lay out our view of the importance of models when used as layers of abstraction for problem-solving and building engineering solutions. Chapter 3 explains the fundamental concepts of power systems operations and control, focusing on hydropower plants. Readers with proficiency in the subject may skip this particular chapter. Chapter 4 discusses previous works related to the development of applications for power systems and similar industrial areas, emphasizing the existing gaps which our research proposes to fill. Chapter 5 describes the D-SPADES MDE approach, including the EMSML language, the model transformation strategy, and the proposed software process. Chapter 6 describes the application of D-SPADES to case studies, as well as results and discussion. Finally, Chapter 7 presents conclusions and discusses directions for future work.

2 CONCEPTUAL BACKGROUND

In this introductory chapter, we present some of the fundamental notions of modeling in science and engineering. We discuss how modeling paradigms are arranged in layers, therefore raising the level of abstraction in which the engineering problems are dealt with. We give an introduction to the roles of modeling in software engineering: the concepts, advantages, and limitations of a model-driven approach to this area. Finally, we point to the potential benefits of an MDE approach to the development of applications in the power systems domain.

2.1 MODELS AND MODELING

Modeling is a central activity in any scientific and engineering area. Models are abstractions of reality, offering approximations. Strictly speaking, it would imply that *all models are wrong, although some are useful*, as quoted by Box and Draper (2007). Therefore, besides those inherent limitations, models have the ability to provide insight and predict the behavior of a process (LEE, 2014). Models are used in place of the real components of a system (physical, logical, or both), to allow a simpler, safer, and cost-effective way to study and design these components. A model can be either: (a) concrete when it is constructed as a material object in the physical world – like a small-scale model of a dam or building; or (b) abstract when any material realization of the model is just incidental – like written mathematical equations representing some phenomena or a diagram drawn in paper. In both science and engineering, the "thing being modeled" is typically an object, process, or system in the physical world, although it could also be another model (LEE, 2016; LEE, 2018).

We convention to call the "thing being modeled" the **target** of the model. In all Engineering fields, the "targets" are constructed in such a way that they emulate the properties of the models. In other words, the models provide the design, while the target is the actual implementation.

The development of technology is strongly influenced and governed by paradigms. A paradigm is *"a conceptual framework that practitioners use, often unknowingly, to interpret observations and develop theories"* (LEE, 2018). Paradigms are often so entangled in our minds that we can't even perceive their existence, yet they shape our understanding of the world around us. Lee (2018) refers to such paradigms as *"unknown knowns"*: knowledge we possess without

being consciously aware. For instance, Newton's second law ($F = m \cdot a$) is a model of an object's motion when subjected to a force. It has meaning within paradigms like the concepts of force, Newton/Leibniz calculus, and the Newtonian notion of time and space, although no physical explanation is attributed to these concepts in classical mechanics[1]. Yet civil engineers are perfectly capable of designing a bridge using classical mechanics: the paradigms related to this field of expertise are part of their "unknown knowns".

In Engineering, every design is a model, which can range, for instance, from a simple sketch of a part to a sophisticated schematic of a complex electrical network. Such models are constructed within modeling frameworks that provide the **syntax** – how it is written down or rendered in physical form, and the **semantics** – the meaning of a given rendition (LEE, 2016). The paradigms of a modeling framework frequently fall into the category of the "unknown knowns".

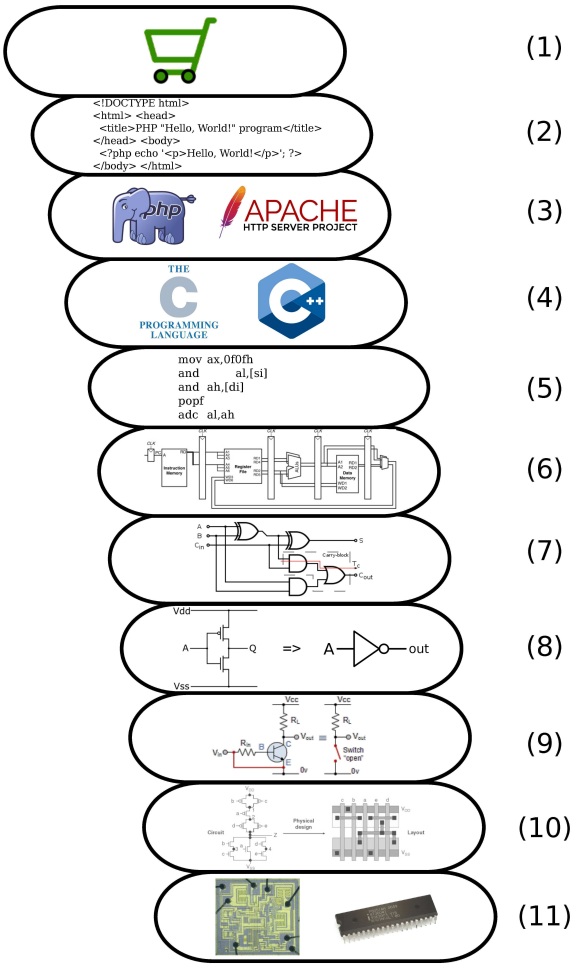
Another interesting property of engineering models is that they stack up in layers, and the design of each layer generally affects only the designs in layers immediately above and below it (LEE, 2018). Let's imagine an example: the e-commerce application schematically represented as a layered perspective in Figure 1. Many layers of modeling exist between the semiconductors of the physical computer and the e-commerce application. Assume the designer decides to use the PHP language to develop this application[2]. He definitely requires knowledge of PHP technology (layer 2). He also requires knowledge of the business logic requirements for the application – the layer above (1), and normally some basic understanding of the runtime platform where the PHP application is deployed – the layer below (3), which normally includes a web server like Apache and associated libraries. The PHP runtime platform though is layered on top of other technologies: the web server and the PHP interpreter module themselves are written in third-generation programming languages like C/C++ (layer 4). But the designer of our e-commerce application requires no knowledge of C/C++ language in order to accomplish his task, yet he is indirectly making use of the technologies on this level of abstraction.

So we have seen that the e-commerce application (1), written in PHP (2), is deployed on the web server (3), written in a third-generation programming language (4). If we keep digging deeper into the layers of Figure 1, we see that programs written in a third-generation programming language (4) are transformed into an assembly language for a given instruction

[1] Sir Isaac Newton rather built a self-consistent and self-referential model where these concepts are defined one in terms of each other.

[2] This example is inspired by the ones given by Lee (2018).

Figure 1 – A conjectural e-commerce application built on top of several layers of modeling paradigms.



Source: The author.

set architecture (5), which through a process called compilation originates executable code that runs in equipment that follows digital machine models (6). Digital machines are capable of performing arithmetic operations and manipulating bits representing text. They are built by associating several blocks of synchronous digital circuits, abstracted as logic diagrams (7). Logic diagrams represent a network of logic functions, each one corresponding to a natural language operation such as “AND”, “OR” and “NOT”, called “logic gates” (8). Logic gates in turn can be modeled as an association of digital switches (9) built using transistors. To produce transistors on a silicon wafer, a layout (10) following the Mead-Conway approach is usually designed. Finally, a digital processor (11) can be built into a chip, where the target of our e-commerce application model (1) can be deployed.

Thus the arithmetic expressions supported by the digital machine become a virtual medium for expressing models, which are translatable into physical manifestations through one level of indirection (LEE, 2018). The models in higher levels also show a transitive relation

to the lower levels, which in other words means all models expressible at a given layer can be translated into the lower layer (but the opposite is usually not true). For instance, a script written in PHP language can be mapped to the set of PHP library functions it calls. In turn, each PHP library function can be mapped into a C procedure and so on. On the other hand, not every C procedure can be mapped back into a PHP script.

Ultimately one could argue that all those eleven layers of modeling paradigms refer to a final realization of the e-commerce application: the interaction between interconnected transistors in a chip. So, theoretically, one could implement our e-commerce application by building a large network of transistors, but that certainly wouldn't be the most productive and flexible way. In a realistic scenario, people write a software program that is translated into a binary pattern that controls a machine composed of a network of transistors.

Consider a less extreme case: imagine that we want to go just two layers down, and develop our e-commerce application directly on a third-generation programming language like C; although perfectly feasible, the required development effort would be considerably larger using plain C language rather than PHP; and the system produced with such an approach would be considerably more complicated to maintain and evolve. So it is inevitable to ask: why PHP appears to be better suited for the development of an e-commerce application than plain C? The PHP project page[3] describes PHP as *"a general-purpose scripting language that is especially suited to server-side web development"*. On the other hand, Kernighan and Ritchie (1978) state that *"C provides no operations to deal directly with composite objects such as character strings, sets, lists or arrays..."*, but the language *"reflects the capabilities of current computers"*; in other words, C provides constructs that map efficiently to typical machine instructions, which operate on bits and bytes. Both languages appear to be well suited for tasks at their respective level of abstraction: PHP development involves heavy manipulation of strings, images, and files, while C programs can efficiently manipulate bits and bytes inside the processor's registers.

The idea is that an engineer in charge of designing and developing a web application is not required to manipulate bits and bytes directly, and certainly doesn't need to possess deep knowledge of semiconductor physics in order to deploy his application on a digital processor. Actually, no single person can possibly master the knowledge to perform all the tasks from top to bottom: each layer of modeling allows individuals to contribute to the design without (necessarily) having knowledge or concern about how the layer of modeling they are creating

[3] https://www.php.net/

will be used by other designers (LEE, 2018) In fact, the developer doesn't need to be aware of any of the intermediate levels of abstraction, except possibly for the ones immediately above and below the PHP language, and still be capable of developing a full-fledged application. Lee (2016) has observed that the choice of modeling framework has profound consequences, since a language suited for a task like 3D modeling is not well suited for modeling the dynamics of an electric circuit, while the paradigms used in the design of such circuits are not the most adequate for producing a computer program. The modeling task becomes more productive when performed at the adequate level of abstraction for the task at hand: in the example from Figure 1, PHP is the right paradigm[4] to model a web application, while the C language is the right paradigm to model the web server and the PHP interpreter module.

In other words, the choice of the appropriate level of abstraction strongly affects how effectively an engineering task can be accomplished. Likewise, modeling an application for a given domain using languages based on paradigms familiar to the specialists of that domain have the potential to increase productivity and product quality. However, we have to point out that there is a price to be paid for designing at higher levels of abstraction:

- The transitivity property implies that at higher levels, fewer possibilities are available to the designer. For instance: the set of operations that can be performed in a PHP script is limited by the functions implemented in the PHP library. There are many other operations that can be performed in the layer below (the C language) that are simply not available in the layer above.

- There is usually a performance cost involved in designing at higher levels of abstraction. For instance: a PHP script that prints the string "Hello, World!" will usually run slower and require more computational resources than its C counterpart.

These limitations however tend to be vastly paid off by design productivity and scalability.

2.2 MODELS IN SOFTWARE ENGINEERING

France and Rumpe (2007) had observed that the process of analyzing a problem, conceiving and expressing a solution in a high-level programming language can be viewed as an implicit form of modeling. Therefore even traditional software development tasks can be classified as model-based problem-solving activities. Writing code is a modeling activity

[4] The "right paradigm" in the sense that it is at the adequate level of abstraction.

because it requires knowledge of the conceptual framework – the paradigms – provided by a programming language. However, as we have discussed above, the paradigms of a specific programming language may not be the most adequate for describing models of a given problem domain. France and Rumpe (2007) argue that the modeling techniques should be more effectively leveraged in software development, and the research question that motivates MDE research should be:

> How can modeling techniques be used to tame the complexity of bridging the gap between the problem domain and the software implementation domain? (FRANCE; RUMPE, 2007), p. 4.

According to France and Rumpe (2007), *a problem-implementation gap exists when a developer implements software solutions to problems using abstractions that are at a lower level than those used to express the problem*. Based on our experience, we could say that most CPS development activities performed today – particularly those related to EMS applications – suffer from the problem-implementation gap: abstractions at a lower level than those used to express the problem are being used to implement the software.

Models with the appropriate level of abstraction are essential for modern software development, as evidenced by four main facts (BRAMBILLA *et al.*, 2017):

1. Software artifacts may be highly complex, and need to be discussed at different abstraction levels, depending on the profile of the involved stakeholder, phase of the development process, and objectives of the project.

2. Software is already widespread in many human activities. The demand for new developments, as well as the maintenance and evolution of existing systems, is expected to increase.

3. The job market experiences a shortage of skilled software development professionals.

4. Software development is not a self-standing activity: interactions between developers and other professionals are often required during the process. Such interactions must be based on a common understanding – a model – of the concepts being discussed.

In this sense, an interesting discussion is made on how models can be used[5] (BRAMBILLA *et al.*, 2017):

[5] This classification is used in Brambilla *et al.* (2017) and credited to Martin Fowler (https://martinfowler.com/bliki/UmlMode.html).

1. **models as sketches:** they specify only partial views of the system, focused on communicating some aspects (like documentation) rather than completeness;

2. **models as blueprints:** they provide a complete and detailed specification/design of the system. As in other fields of engineering, such designs can be handed off to a separate group to write the code, much as blueprints are used in civil or mechanical projects.

3. **models as programs:** models are used as the primary development tool, instead of lower-level programming languages. Tools can be used to transform the models and compile them into executable code.

According to France and Rumpe (2007) a general perception exists in that development models are primarily documentation artifacts, and thus they are peripheral to software development. This perception limits the use of high-level models to activity (1) above. However, during the development process, models can be used in all the manners specified above. For instance, sketches of a system can be used in early project discussions and design decisions (1); afterward, complete models can be defined as system blueprints (2); and finally, the same blueprints can be further refined to produce software artifacts (3) through transformations and code generation.

2.2.1 No Silver Bullet

Brooks Jr (1987) observes a distinction between two types of complexity that usually emerge in software systems: “accidental” and “essential” complexity.

1. In **essence**, a software entity is an abstract construct of interlocking concepts, which remains the same no matter the representation used (programming paradigm or language); this complexity is inherent to the nature of software.

2. By **accident** some difficulties are observed in the production of software, but those are not inherent; such difficulties are eliminated, for instance, by using high-level programming languages which abstract the computer hardware and let the programmer concentrate on the program construct.

Brooks Jr (1987) argues that advances like third-generation programming languages and object-oriented programming can do no more than remove the accidental difficulties from the expression of the design. The complexity of the design itself is essential and cannot be eliminated by such techniques. Therefore, there is no “silver bullet”:

> There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity (BROOKS JR, 1987), p. 2.

Although no technological leap or "silver bullet" is claimed by the current research, we argue that developing software for a given domain of application can be much more productive when performed using specialized tools and methodologies for that particular domain. For instance, the methodologies and tools that are great for developing an e-commerce application (javascript, HTML, CSS, etc) are certainly not the most appropriate for an embedded real-time controller, which in turn could be better developed using specialized development environments, languages, and associated libraries.

The role of MDE, as already mentioned above, is *to promote technologies that support the systematic transformation of problem-level abstractions to software implementations, thus reducing the gap between problem analysis and software implementation domains* (FRANCE; RUMPE, 2007). So let's imagine an engineer dealing with the complexities of managing the operations of a large power plant or interconnected power system: what is the complexity of program data structures, loops, and branches other than accidental for this engineer? Although we do not claim to be able to increase productivity by one order of magnitude, we believe the MDE approach may help dealing, not just with the accidental complexity, but also with essential complexity, by re-using physical and business logic models already developed for other phases of the CPS design.

2.2.2 Advantages of the MDE Approach to EMS Applications

Below we list some of the expected advantages of applying the MDE approach to EMS applications development:

- Re-use, at least partially, the models described for planning, simulation, and validation in the electromechanical domain for actual software development, instead of using them only as input for the requirements elicitation phase of the software engineering process.
- Since models can be verified and validated, design correctness is assured at an early stage, even before other software artifacts are produced. Therefore, the approach can reduce the number of software defects due to design errors.

- Overall software quality is improved by applying a well-defined and highly automated process, thus reducing defects due to implementation errors.
- Aspects like the consistency of implementation with respect to the design and requirements traceability are enforced.
- Reduce the cost and effort involved in the procurement of SCADA/EMS. According to (STRASSER *et al.*, 2009) the software engineering process represents up to 80% of the total cost of such systems.

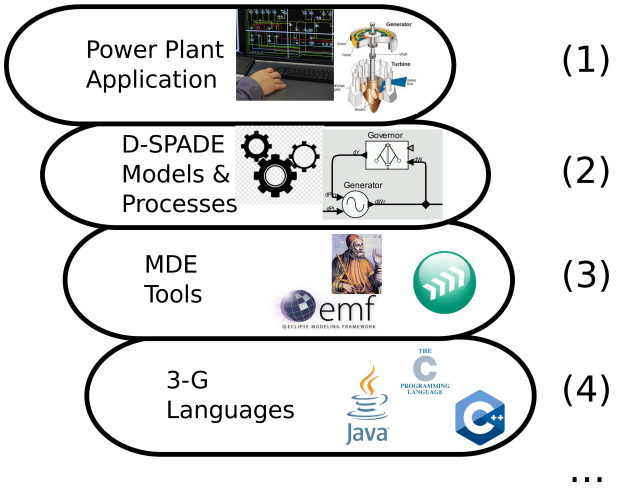
2.2.3 Situating our Proposed Approach

The current EMS applications design process makes use of models only as sketches or blueprints. In this work, we propose D-SPADES: a domain-specific approach to the engineering of EMS applications. With this approach, the modeling activities are promoted to a central role in the development process, offering a more appropriate layer of abstraction for describing the software solutions. Figure 2 schematically represents the layers of paradigms involved in the development of a power plant application using D-SPADES. Notice that instead of writing the applications directly in traditional third-generation programming languages, the D-SPADES approach adds additional layers of paradigms: the use of domain-specific models and MDE tools. We believe that a model-driven approach based on paradigms familiar to the specialists from the power systems sector will help cope with some of the challenges mentioned in Chapter 1, specifically: the development, maintenance, and evolution of customized EMS applications in a scenario where the layers of abstraction supporting these applications, i.e. the SCADA software and hardware platforms, may change frequently. It is relevant to mention that these additional layers of abstractions, and the corresponding run-time overhead introduced, must not hinder the timing constraints of the EMS applications. Although the focus of this work is not on real-time software design methodology, aspects related to timing performance are taken into account.

2.3 MODELING PARADIGMS, LANGUAGES AND TOOLS

A “modeling paradigm” can be defined as *a set of requirements that governs how systems within the domain are to be modeled* (NORDSTROM *et al.*, 1999). In a simpler phrasing, as briefly discussed in Section 2.1, a paradigm is *“a conceptual framework used to interpret*

Figure 2 – Schematic representation of the layers of modeling paradigms in D-SPADES.



Source: The author.

observations and develop theories" (LEE, 2018). Nordstrom *et al.* (1999) also states that *the modeling paradigm defines the language for modeling systems in the domain*. Therefore a modeling paradigm allows us to, at least:

- represent/model some phenomenon or system through a language;
- interpret observations and develop theories using such models.

In the following section we discuss a modeling paradigm we believe is particularly relevant to this work, as well as some modeling tools that can support this paradigm.

2.3.1 Actor-Oriented Programming Models

Actor-based or actor-oriented (Actor-Oriented Programming Model (AO)) programming model (LEE, 2003) is a general-purpose concurrent programming model with wide applicability. It can target both shared- and distributed-memory architectures, facilitating geographical distribution, and providing strong support for fault tolerance and resilience (BUTCHER, 2014), although it can also be deployed in non-distributed systems. According to Agha (1990), AO models have a flexible structure, particularly well-suited for rapid prototyping applications.

The concept of AO used here is described in the work of Prof. Edward Ashford Lee and his group with the *Ptolemy II* project at UC Berkeley (LEE, 2003; LEE, 2014). That notion in turn is credited to the work of Agha (1990) and others. The term "actor" expresses the concept of a self-contained, interactive, and independent component of a computing system

that communicates by asynchronous message passing (AGHA, 1990)[6]. The behavior of such a component is therefore triggered by incoming messages. The triggering by message passing contrasts with the traditional view of abstract data structures interacting via procedure call (or method invocation) in OO and structured programming paradigms (INDRUSIAK; GLESNER, 2006).

2.3.2 Characteristics of AO Models and Design Environments

Tools supporting the AO paradigm often have a block diagram based design environment, where the actual development basically involves assembling preexisting components – the actors – from a library. In such an environment, the concept of an actor is materialized as *an encapsulation of parameterized actions performed on input data to produce output data* (ZHOU *et al.*, 2007). Different compositions of the same actors can implement different functionality.

The actors' interface consists of ports and parameters. Input and output data are communicated through ports, while the internal state and behavior of each actor are hidden from other actors. AO models also contain explicit communication channels that pass data from one port to another. The actors in such models do not interact directly, but instead, use the channels to which they are connected as a means for communication (LEE *et al.*, 2002). The concepts of models, actors, ports, parameters, and channels describe the abstract syntax of an actor-oriented language.

Some examples of widely known software tools supporting AO design include: *Simulink* from *The MathWorks*[7], extensively used in both continuous and discrete time control systems engineering; *LabVIEW* from *National Instruments*[8], commonly used for data acquisition, instrument control, and industrial automation; *The Generic Modeling Environment (GME)* from Vanderbilt University[9], a configurable toolkit for creating domain-specific modeling and program synthesis environments; Several *Modelica Simulation Environments and Libraries*[10] available both commercially and as open source projects, intended for modeling and simulation of complex physical systems containing mechanical, electrical, electronic, hydraulic, thermal, control,

[6] The concept of "actor" in AO has a very distinct interpretation from the one used in OMG's UML, in which "*an Actor models a type of role played by an entity that interacts with the subject*" (The Object Management Group, 2007).

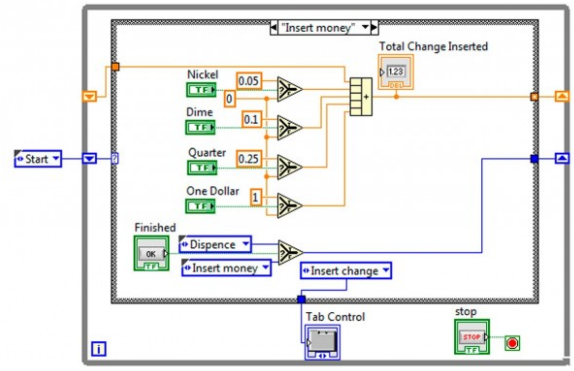
[7] https://www.mathworks.com/products/simulink.html

[8] https://www.ni.com/en-us/shop/labview.html

[9] https://www.isis.vanderbilt.edu/Projects/gme/

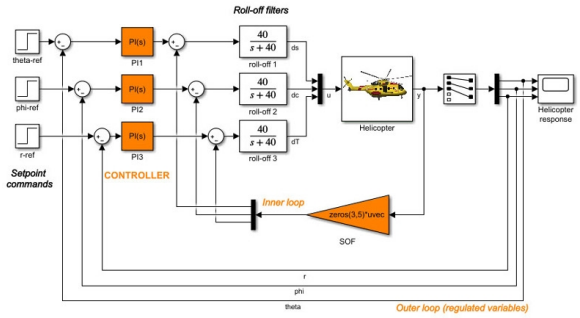
[10] https://www.modelica.org

Figure 3 – Some examples of AO models from different modeling tools.



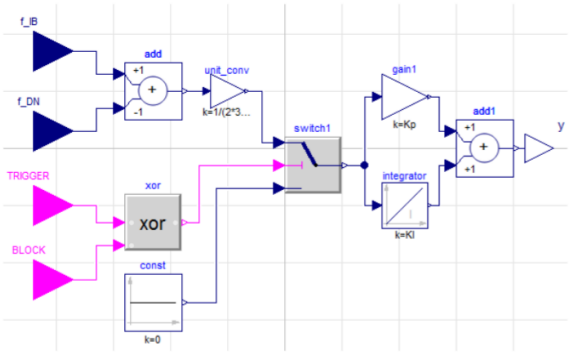
(a) LabView model

Source: https://microcontrollerslab.com/list-labview-tutorials-projects/



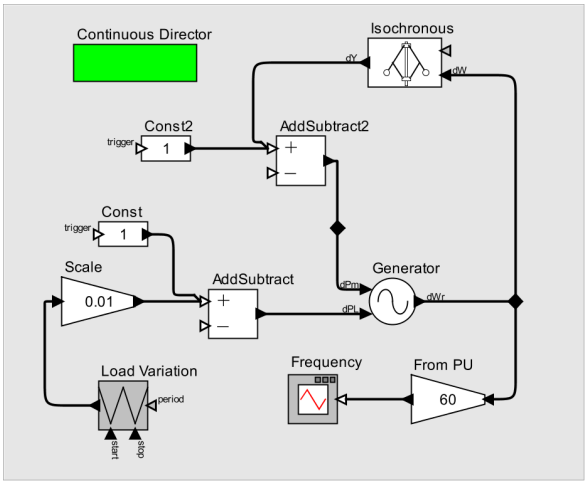
(b) Simulink model

Source: https://www.mathworks.com/products/simulink.html



(c) Modelica model

Source: Mukherjee and Vanfretti (2019)



(d) Ptolemy II model

Source: The author

electric power, or process-oriented subcomponents; and the *Ptolemy II*, from *UC Berkeley*[11], an open-source software framework based on AO design, supporting modeling and simulation of heterogeneous models of computation. Figure 3 shows the appearance of models developed in some of these environments.

The models depicted in Figure 3 intuitively convey the idea that the connections between blocks represent interactions between components in a design. However, the type of interaction is not explicit, for instance, we cannot directly determine whether:

- Is the communication performed in a rendezvous style, as in a phone call, or is it through asynchronous messages, like sending a letter?
- Is it a clocked update of data, as in a synchronous digital circuit?

[11] https://ptolemy.berkeley.edu/

- Does time play a role in the interaction?
- Is the interaction discrete or continuous?

As we can see, the *syntactic structure of an actor-oriented design says little about its semantics* (LEE, 2003). The actual semantic – what the model means, and what it does – is determined by the Model of Computation (MoC) implemented by the modeling tool. The MoC dictates the operational rules for executing a model. These rules determine when actors perform internal computations, update their internal state, and communicate with other actors. The MoC also defines the nature of the communication (LEE, 2003). In other words, one single syntactical representation – for instance, the one in Figure 3 (d) – may have significantly different semantics, i.e. the model works differently, depending on the MoC under which it is executed.

Some possible examples of MoC that may govern the execution of AO models include:

- **Synchronous Dataflow Model of Computation (SDF)**, also called *static dataflow* (LEE *et al.*, 2014), is a specific type of dataflow model in which actors begin execution (they are fired) when their required data inputs become available. When executed in a sequential computer, the actors' order of execution is determined by a fixed schedule of "firings" by the SDF scheduler.
- **Process Network Model of Computation (PN)** is a MoC in which each actor runs concurrently in its own thread of execution (SMYTH *et al.*, 2014). That is, instead of being explicitly scheduled (fired) by a scheduler, PN actors are defined by a (typically non-terminating) program that continuously reads data tokens from input ports and writes data tokens to output ports. Conceptually, all actors execute simultaneously, although the actual computational platform may involve the use of shared resources by interleaving tasks.
- The **Continuous-Time Model of Computation (CT)** conceptually models time as a continuum (CARDOSO *et al.*, 2014). The continuous dynamics of physical processes are represented using Ordinary Differential Equations (ODEs) over a time variable. The model's behavior with respect to time – the inputs and outputs of each actor – is determined by numerically solving a system of ordinary differential equations.

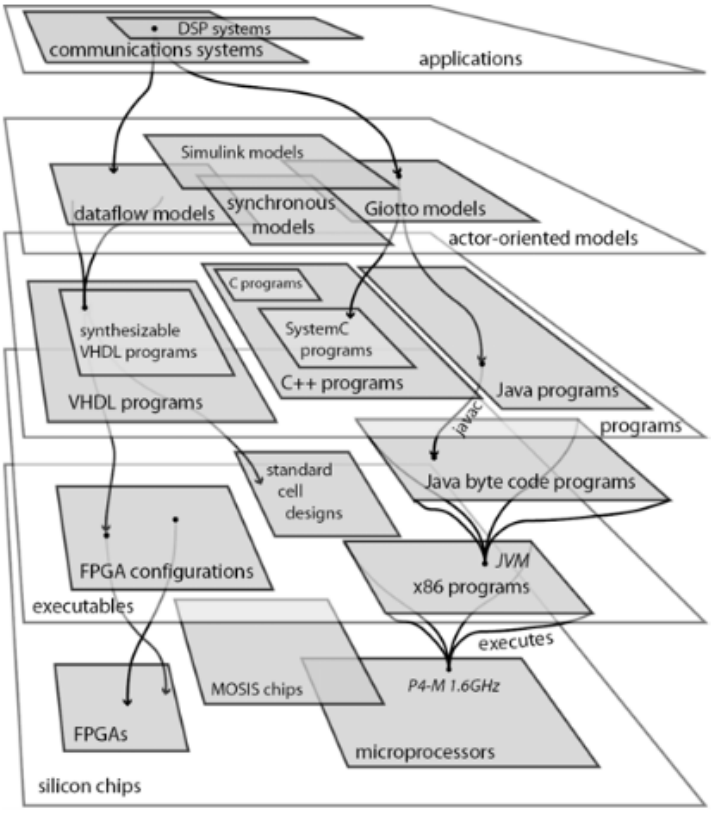
Each software tool capable of AO design may support different sets of MoCs. The *Simulink* tool, for instance, supports a fixed CT MoC, in which discrete-time signals are supported

as a special case of continuous time, by making the discrete signal piecewise constant in continuous time (ZHOU *et al.*, 2007). In comparison, the *Ptolemy II* platform isn't limited to a single built-in MoC, but instead supports multiple *semantic domains*, which is the term used to refer to the implementation of a MoC (LEE, 2014). This is one of the main distinguishing features of *Ptolemy II* in comparison to other design environments.

2.3.3 Advantages of the AO Model

The concept of platforms, platform-based and model-based design is discussed by Lee *et al.* (2002). Lee (2003) argues that *model-based design and platform-based design are essentially two sides of the same coin*. The model-based design consists in specifying designs using *platforms with useful modeling properties*. Figure 4 schematically represents some platforms and their interrelationships.

Figure 4 – An illustration of some platforms and their interrelationships.



Source: Lee (2003).

In the conceptual layering of Figure 4, a control system designed with a tool like, for instance, *Simulink* stands in the "actor-oriented" platform, which is one level above Third-generation Programming Language (3GL) programs written in, for instance, C++. Let's consider

a case where the same control system is to be designed using languages such as C++ or Java, and represented by abstractions such as UML. In this case, object-orientated components interact with one another essentially by method calls, which represent a transfer of control. The concurrent execution of tasks in these platforms is managed through complex low-level abstractions such as threads, mutexes, and semaphores. The control systems specialist would need, besides knowledge of his domain's own paradigms, familiarity with concurrent programming paradigms and languages. It would be much more appropriate for this control system's specialist to be able to convey his design using *platforms with modeling properties that reflect requirements of the application, not accidental properties of the implementation* (LEE, 2003). Managing concurrency, passing the correct types in the method invocation, and handling exceptions are clearly accidental complexities of the programming platform.

In actor-oriented abstractions, the low-level mechanisms of concurrent programming are abstracted from the designer. They can be considered as "*assembly-level mechanisms*" (LEE, 2003), from the AO platform's point of view. Threads and mutexes become implementation mechanisms instead of part of the programming model (ZHOU *et al.*, 2007). Besides, AO models are much more intuitive for domain specialists, for instance, from the power systems area, which are inherently familiar with design strategies based on block diagrams and transfer functions. Lee *et al.* (2002) point out that *perhaps the most significant advantage of actor-oriented design is the use of patterns of component interaction with useful modeling properties*, which consist in the different MoCs.

As we will further discuss in Chapter 4, many works in the power systems' area adopt modeling paradigms borrowed from control theory, namely the block diagram and transfer function notation, which are compatible with the AO paradigm. Modeling languages associated with traditional modeling tools for Science and Engineering are also known to be used in this area, for instance: Modelica, Matlab/Simulink, Scilab and *Ptolemy II* are often used in power systems modeling, as described in references (SULLIGOI *et al.*, 2011; SÜSS *et al.*, 2008; ZHABELOVA *et al.*, 2014; ZANABRIA *et al.*, 2016; STIFTER *et al.*, 2013; BOGODOROVA *et al.*, 2013; ZHABELOVA *et al.*, 2014). These modeling environments integrate a visual schematic and equation-based notation, which fits the description of AO design. Our review has also revealed the use of the languages specified in the IEC 61499/61131 standards for distributed control applications, particularly in smart-grids-related areas, according to references (ANDRÉN *et al.*, 2013; ANDRÉN *et al.*, 2014; ANDRÉN *et al.*, 2013; ZHABELOVA *et al.*, 2014; STIFTER *et*

al., 2013; STRASSER *et al.*, 2014). IEC 61499 specifies a language based on a "function block" concept. The standard also specifies an event-oriented model of computation and therefore can be considered compatible with the AO paradigm. In this work, we propose adopting an actor-oriented modeling paradigm for developing EMS applications. The following section provides a brief overview of the *Ptolemy II* tool, which supports the AO paradigm and is used in the remainder of this work.

2.3.4 The *Ptolemy II* Tool

Ptolemy II is a software modeling tool suited for modeling, simulation, and design of concurrent, real-time, and embedded systems. The tool's design environment is focused on the assembly of components (i.e., actors) from a library, which operates according to a chosen MoC that govern the interaction between components (ZHOU *et al.*, 2007). *Ptolemy II* is heavily based on the concept of actor-oriented models discussed above, in which actors execute concurrently and transfer data to each other via ports (BROOKS *et al.*, 2014a).

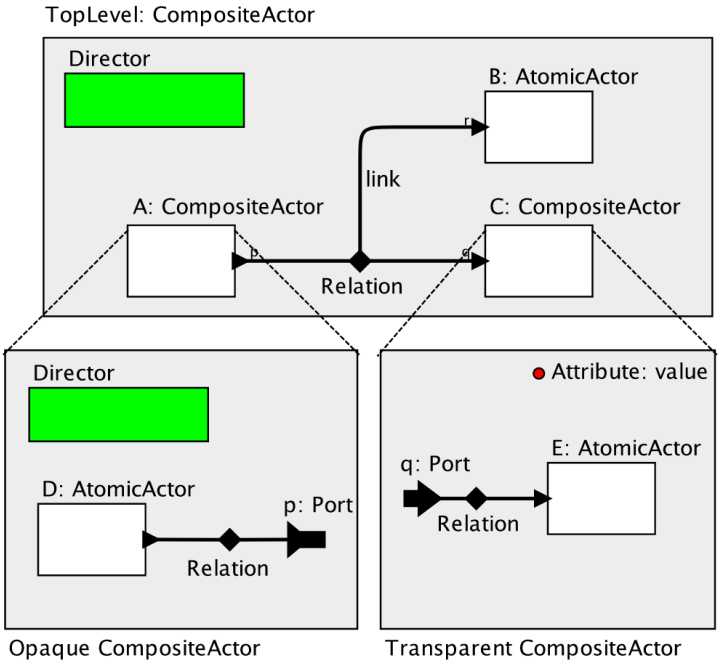
In terms of software, *Ptolemy II* is implemented as a set of Java packages, roughly composed of (BROOKS *et al.*, 2014a): a **kernel package** that supports clustered hierarchical graphs, which are collections of entities and relations between those entities[12]; the **data package** defining the classes that carry data from one component to another in a model; an **actor package** that extends the kernel, so that entities have functionality and can communicate through ports and relations; The actor package also includes the base definition of a *Director* class, which is the concept used to specify the **domain** of a given model, i.e., the desired MoC that controls the execution of the model.

The abstract syntax of *Ptolemy II* models consists in a **tree**, which represents the hierarchy of models, overlaid with a **graph** at each level of the hierarchy; the graph specifies the connections between components of a model (BROOKS *et al.*, 2014a). Figure 5 schematically represents an example of such a hierarchical model using a visual notation similar to the actual graphical concrete syntax of the *Ptolemy II* environment.

In the *Ptolemy II* syntax, a model essentially consists of a top-level entity that contains other entities. The entities – or actors – have ports through which they interact and exchange data with other entities. Their interactions are mediated by relations, which represent communication

[12] Note on the use of the term "Entity" vs "Actor": in terms of implementation, an entity is a more generic type. An actor is a subclass of entity; a state inside an Finite State Machine (FSM) is also a subclass of "Entity".

Figure 5 – *Ptolemy II* Hierarchical Model Structure.



Source: (BROOKS *et al.*, 2014a).

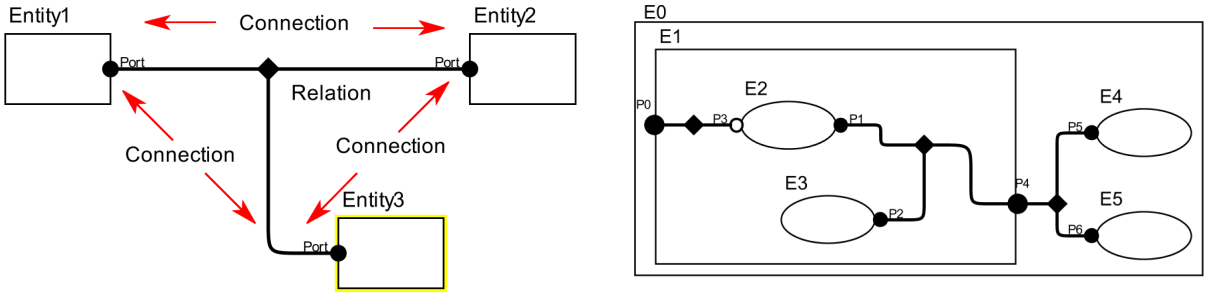
paths. All of these objects (entities, ports, and relations) can have attributes, which define their parameters and add annotations. Ports have links to relations, represented in the meta-model as an association between the Relation class and the Port class (BROOKS *et al.*, 2014a).

In *Ptolemy II* an actor can itself be a model, therefore being referred to as a "composite actor". A composite actor that contains a director (depicted as a green rectangle in the models from Figure 5) is said to be opaque; otherwise, it is transparent. An opaque composite actor behaves like a non-composite (i.e., atomic) actor and its internal structure is not visible to the model in which it is used; it is a black box. In contrast, a transparent composite actor is fully visible from the outside, but is not executable on its own (BROOKS *et al.*, 2014a), since it does not have an associated MoC.

The concept reciprocal to the "composite actor" is the "atomic actor", whose terminology is derived from the Greek word *atomos*, or indivisible (LEE; MESSERSCHMITT, 1987). Atomic actors are not composed of other actors and have to be implemented directly on the underlying platform's programming model, which in the case of *Ptolemy II* is the Java language.

The language used for expressing *Ptolemy II* models is called "XML Modeling Markup Language (MoML)". MoML specifies *interconnections of parameterized, hierarchical components, while making no assumptions about the meaning of the components or their interconnections* (LEE; NEUENDORFFER, 2000). A model in MoML is represented as a clustered graph,

Figure 6 – MoML notation, terminology and aggregation of sub-models.



(a) MoML visual notation and terminology **(b) Composite models: aggregation**

Source: The Author, adapted from Lee and Neuendorffer (2000).

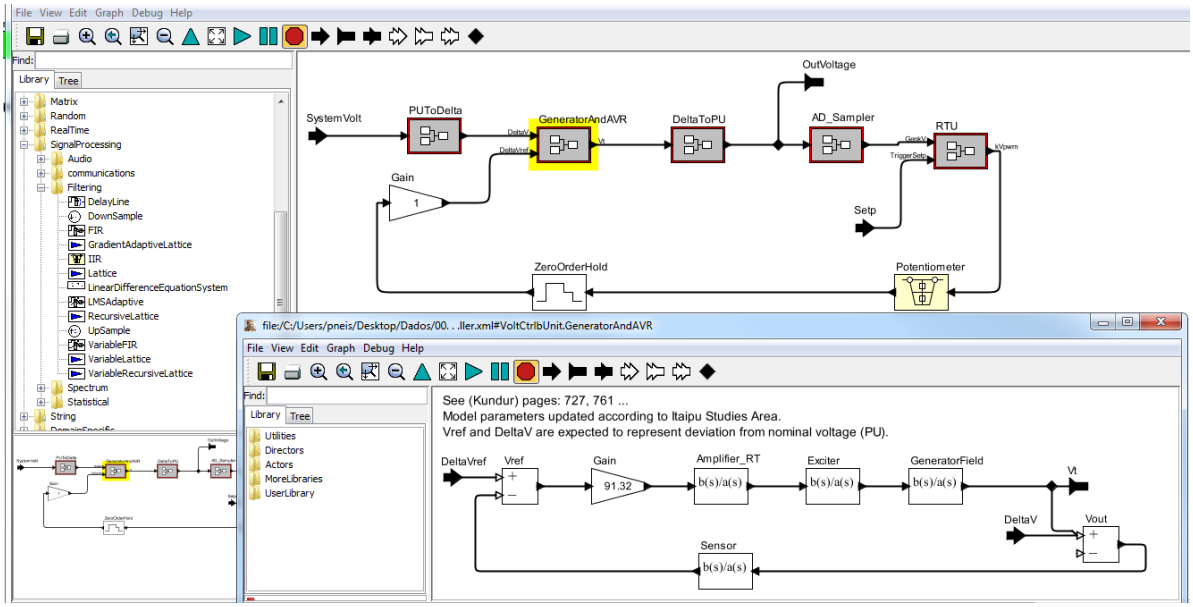
and it can convey designs such as netlists, state transition diagrams, and block diagrams, among others. In MoML terminology, a **topology** is a collection of entities, ports, and relations. Figure 6 (a) shows a graphical notation representing a MoML model. In this figure, **entities** are depicted as boxes and **relations** as diamonds. Entities contain **ports**, shown as filled circles, and relations connect ports through **links**. **Composite entities** are clusters containing another topology, like conceptually illustrated Figure 6 (b), where **atomic entities** are represented as circles in order to differentiate them from composite entities[13].

Ptolemy II has the *Vergil* graphical editor; it can be used to create, manipulate and execute AO models. This graphical editor has the appearance shown in Figure 7. This image shows two Vergil windows: the background window contains a model of one of the Itaipu generating unit's voltage control system, and a foreground window shows the contents of one of the composite actors from the background model – the yellow highlighted labeled as "GeneratorAndAVR". The "GeneratorAndAVR" composite actor represents a particular case of the feedback control model for the generating unit's excitation system discussed in Chapter 3, and shown in Figure 16 (see page 55). Some features of a typical AO design environment can be pointed out in the Vergil editor: the block-diagram-oriented design, and the actors' library (on the left-hand side), which can be used to assemble new models from preexisting components.

The design shown in Figure 7 is a *clustered graph* expressed in a concrete syntax of the MoML language, like conceptually represented in Figure 6. This concrete syntax has some visual differences/augmentations, for instance: actors may have an individual pictorial representation (an icon) associated; ports are represented as arrow points; links are shown as

[13] The structural relationship between *entities*, *composite entities* and *atomic entities* is identical to that observed in the traditional *composite design pattern* (GAMMA *et al.*, 1995) from object-oriented design. The implementation of both *Ptolemy II* and D-SPADES use the composite pattern as the realization of this structure.

Figure 7 – *Ptolemy II* Vergil editor window.



Source: The Author.

routed lines connecting ports and relations; relations are represented as filled diamonds, but are not necessarily displayed unless more than two ports are linked together. The Vergil editor serializes (saves) these models to XML files, which consist of an alternate concrete syntax for the MoML language. These AO models can be described by a metamodel, and therefore are suitable to be used in model transformations such as M2T.

2.4 CHAPTER SUMMARY

In this chapter, we discussed how models are stacked up as platforms in order to raise the level of abstraction and increase the productivity of engineering tasks. In this scenario, we contextualize the D-SPADES approach, arguing that it offers an appropriate layer of abstraction for describing EMS software. We acknowledge that although “no silver bullet” exists such that productivity increases tenfold, a domain-specific approach is leveraged by the paradigms familiar to the professionals from the power systems domain – their “unknown knowns”. We have also briefly discussed the actor-oriented modeling paradigm, which offers a good notation, compatible with abstractions traditionally used for expressing problems in the power systems’ domain. This paradigm is supported by some well-known modeling and simulation tools, however, a fully integrated MDE approach applicable to the development of EMS applications, based on such language, is still absent. In the forthcoming chapters, we propose an approach for developing EMS applications based on the AO paradigm and model transformations.

3 BACKGROUND ON THE APPLICATION DOMAIN

This chapter describes some notions and a basic description of the physical processes involved in the operation of hydropower plants, as well as the interfaces with its digital control system: the cyber part of the CPS. We briefly discuss this domain of application and some of the common paradigms in the area. This overview provides the reader with a basic knowledge of the processes where the software under study is applied. We do not intend to cover in depth the power system's physical models and dynamics. For a more detailed explanation of the theme, the reader shall refer to (KUNDUR *et al.*, 1994; GRIGSBY, 2007; BEVRANI, 2008; EREMIA; SHAHIDEHPOUR, 2013; KOSOW, 2009). Some of the topics covered include: a typical hydropower plant at a glance; the main components of the hydraulic turbine, generator, and speed-governing actuators; the main components of the generator's excitation circuit and its controls; basic speed/frequency and voltage regulation concepts; some of the controls involved in power plant operations; and a particular application scenario at the Itaipu Power Plant.

3.1 A TYPICAL HYDRO POWER PLANT

In Figure 8 we show a panoramic view of the Itaipu Power Plant to illustrate some of the main components of a typical hydropower plant. The highlighted components are:

1. **The Reservoir**: is the storage space for the main body of water, typically created by the construction of the dam. A reservoir needs to be deep enough to create a head[1] of water for the turbines. Some hydropower plants are said to have little or no reservoir: the so-called "run-of-the-river" hydro plants, usually built in a steep valley with constant river flow.

2. **The Dam**: is usually a bulk civil structure, responsible for holding back the water creating the reservoir.

3. **The powerhouse**: is the structure that provides housing for electromechanical equipment like turbines and generators.

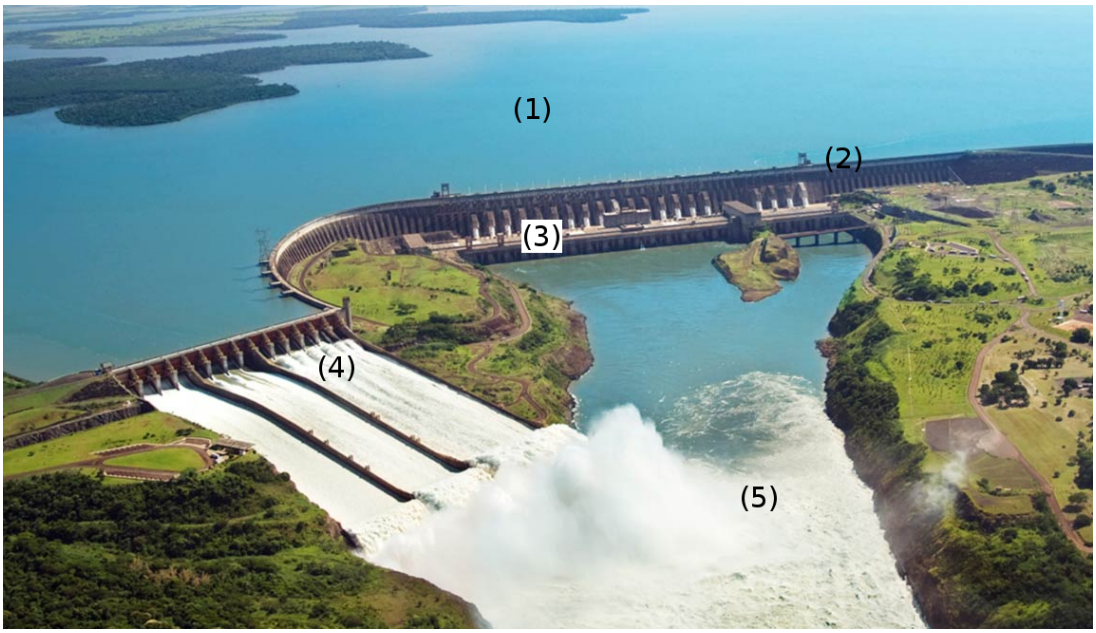
4. **The Spillway:** is a structure that provides controlled release of flows from the reservoir into a downstream area. Excess water not utilized for power production is discharged

[1] Hydraulic head or piezometric head is a specific measurement of liquid pressure, expressed in units of length. The gross head is numerically equal to the difference between the reservoir level and downstream tailrace level.

through the spillway, especially during high inflow season, so the dam is not overflown. In other words, spillway operation helps regulate reservoir level, total downstream flows, and consequently downriver water levels. Proper spillway operation prevents uncontrolled floods, both upstream and downstream of the dam. It can also influence navigational conditions in the associated river system.

5. **The tailrace**: is a channel that carries water away from a hydroelectric plant. The water in this channel has already been used to rotate the turbines and produce power. The tail race is usually at a much lower level than the height of the reservoir behind the dam, and this difference – the gross head –, along with the volume of water flowing, determines the amount of power that can be obtained from the water.

Figure 8 – Overview of a typical hydro power plant, in this case Itaipu.



Source: Itaipu Power Plant.

3.2 OVERVIEW OF THE HYDRO POWER PRODUCTION PROCESS

The energy production process by means of hydraulic turbines involves the conversion of the stored energy of a fluid mass (a combination of potential energy and kinetic energy) into mechanical energy (SUBRAMANYA, 2013). Normally a hydroelectric generator, connected to the turbine by means of a shaft, converts this mechanical energy into electricity. Figure 9

illustrates the main components of a hydraulic generating unit. In reaction turbines[2], like that of Figure 9, the process works by forcing the water into a *scroll case* (1), usually through a penstock. Inside the scroll case, water is guided by the *stay vane ring* (2) into the *wicket gates* (3), which are composed of movable parts responsible for controlling the water flow through the turbine blades. The wicket gate works like a controlled valve, being opened and closed by the movement of the *bull ring mechanism* (4). The actuator responsible for turning the bull ring is the *gate servomotor* (5), which is a hydraulic piston. The *turbine* (6) is the actual component where the energy conversion of the falling water into motion takes place. The turbine is connected by means of a *shaft* (7) to the generator's *rotor* (8), composed of rotating electromagnets responsible for producing the magnetic field. The *stator* (9) is the structure holding the armature coils where, guided by Faraday's law, the conversion of movement into electricity occurs. The electricity generated in the stator coils is conducted to the power grid by means of the *bus ducts* (14), usually composed of isolated copper bars. The electromagnets field intensity is controlled by the excitation current injected by the exciter through the *slip rings* (11). The rotating parts (turbine, shaft, and rotor) are mechanically supported by the *thrust bearing* (12), which vertically holds the weight of the whole set. Other components highlighted in the illustration are: the *oil head* [3] (10), the *stator cooling* radiators (13), and the *DC bus* (16) that feeds the exciter and other auxiliary systems[4].

Out of the components briefly described above, the ones closely related to controlling the energy conversion process are:

- **The gate servomotor** (5), which controls the active power and frequency delivered by the generator to the power system. Other important control components, mainly the **speed governor** and the **load-frequency regulation** loops work associated with the servomotors.
- **The slip rings and rotor coils** (11, 8), by means of which the *exciter* and the secondary voltage regulation loops control the generator's voltage and reactive power output.

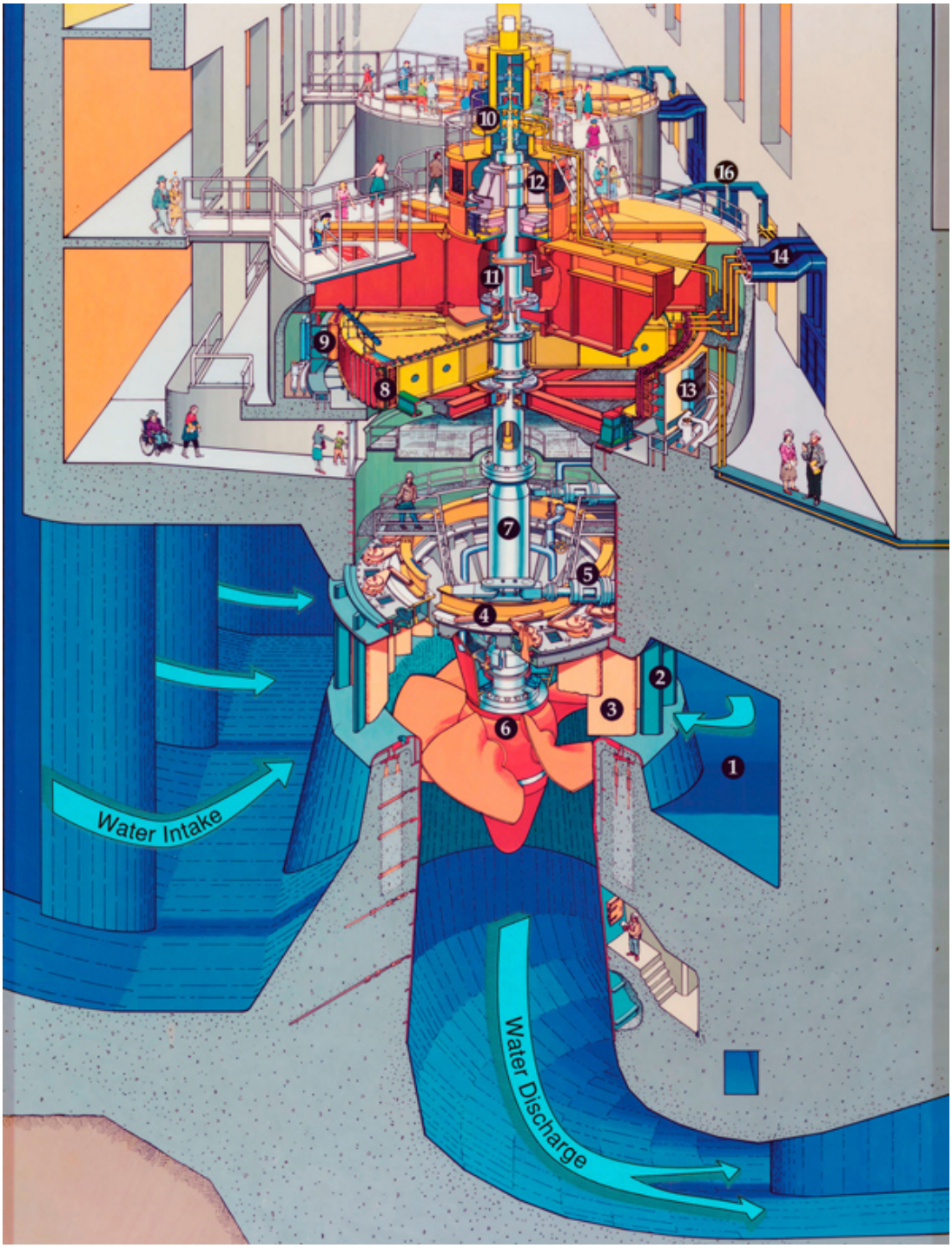
The flows of active and reactive power in the transmission network are fairly independent of each other in the sense that they can be controlled separately, by means of independent control

[2] Regarding the nature of its interaction with water, turbines are classified as (1) *reaction turbines*, where the water pressure changes while flowing through the rotor; and (2) *impulse turbines*, where the water pressure does not change while flowing through the rotor, and the interaction occurs at atmospheric pressure. For more details, the reader shall refer to (KUNDUR *et al.*, 1994; SUBRAMANYA, 2013).

[3] This component only applies to turbines having adjustable blades, like the Kaplan or certain types of bulb turbines.

[4] This image is an adaptation of the publicly available version provided by the US Army Corps of Engineers - http://www.nwp.usace.army.mil/hydropower/

Figure 9 – Hydraulic turbine and generator.



Source: the US Army Corps of Engineers - http://www.nwp.usace.army.mil/hydropower/.

actions. Active power is closely related to frequency control, whilst reactive power is linked to voltage control. Active power and frequency control acts upon the generating units' prime movers, via wicket gate open/close movements. Reactive power and voltage control act upon the alternators' excitation current. Both these control actions are vital for the satisfactory performance of the power system (KUNDUR *et al.*, 1994).

3.2.1 Speed, Frequency and Active Power Regulation

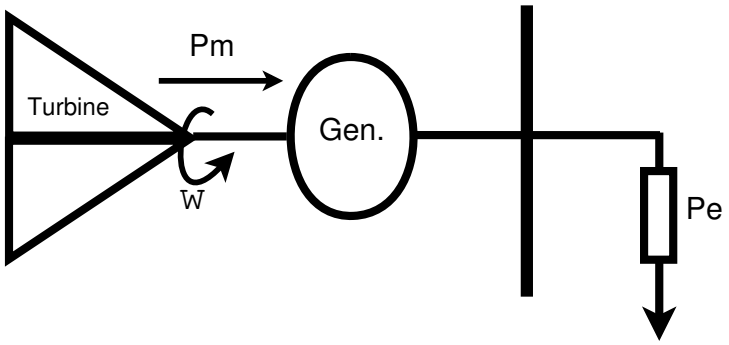
In synchronous generators, the turbine shaft is rigidly connected to the rotating electromagnets responsible for inducing voltages at the armature windings. The frequency of the alternate voltage produced at the generator's terminals is determined by rotational speed and its number of poles P, according to Equation 1 (KLEMPNER; KERSZENBAUM, 2004). Since the number of poles is fixed for a given machine, frequency is locked (or synchronized) to rotational speed. Therefore, in order to control frequency, the prime mover's speed needs to be acted upon.

$$f_{Hz} = \frac{\frac{P}{2} \times \omega_{rpm}}{60} \tag{1}$$

3.2.1.1 Prime Mover, Generator and Load: Frequency Deviation

Figure 10 is a schematic representation of a basic single-generator and load system, in which we assume no speed regulation is performed.

Figure 10 – Schematic representation of turbine, generator and load.



Source: The author.

For this hypothetical system, the initial condition is established such that the turbine mechanical power (P_m) and the electrical load power (P_e) are equal, and the system operates at the nominal frequency. Whenever a small change in electrical load occurs (such as when a light is turned on), with mechanical power remaining constant, the turbine speed (ω) changes according to the rotational inertia (J) of the system. This variation can be modeled by the differential Equation 2 (KUNDUR *et al.*, 1994):

$$P_m - P_e = J\frac{d\omega}{dt} \tag{2}$$

The rotating inertia (J) initially provides the extra energy supplied to the load, at the expense of reducing the rotating speed (and frequency) of the system. The load itself is usually composed of a mix of resistive and reactive loads. Loads composed of electrical motors tend to be dependent on frequency in the sense that their power decreases as frequency drops, whilst resistive loads such as heating are independent of frequency. The overall load's response to the system's frequency can be expressed as:

$$\Delta P_e = \Delta P_L + D\Delta\omega \tag{3}$$

Where ΔP_L corresponds to the fraction of load that is independent of frequency. The $D\Delta\omega$ part corresponds to the frequency-dependent loads, with D being called the "load-damping constant". D is expressed as a percent change in load for a percent change in frequency. Typical values of D are between 1 to 2 percent.

The response of the hypothetical system of Figure 10 is determined by the inertia constant J and the load damping factor D. The steady-state speed deviation is such that the small changes in load are compensated by a correspondent load variation due to frequency sensitivity. In other words, the system remains stable at a different operation condition, i.e. at a frequency slightly different from the nominal value.

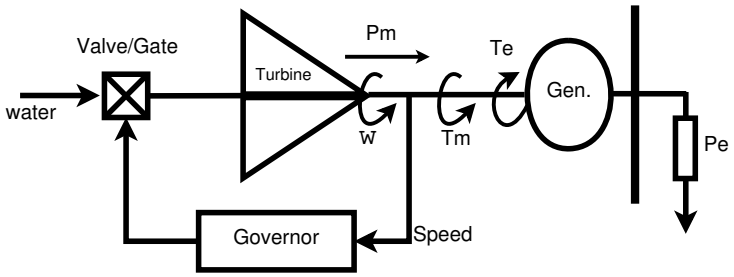
3.2.1.2 Primary Regulation: the Speed Governor

In order to maintain frequency as close as possible to its nominal (or scheduled) value, a *speed governor* is added to the system, as illustrated in Figure 11. The speed governor "measures" the generator's speed and automatically compensates for variations by acting upon the prime mover's valve or wicket gate position. The strategy for compensating speed variations depends on the power network the generator is connected to, and can be either "Isochoronous Speed Control" or "Droop Speed Control".

In **Isochoronous Speed Control** mode[5], the energy being admitted to the prime mover is tightly regulated in response to changes in load which would tend to cause changes in frequency. Any increase in load would tend to cause the frequency to decrease, but energy is quickly admitted to the prime mover to maintain the frequency constant. The system responds likewise for a decrease in load. The isochoronous governor, by definition, has zero steady-state speed error for load variations inside the designed range.

[5] The term Isochoronous means at constant speed (KUNDUR *et al.*, 1994).

Figure 11 – Schematic representation of turbine, generator, load and governor.

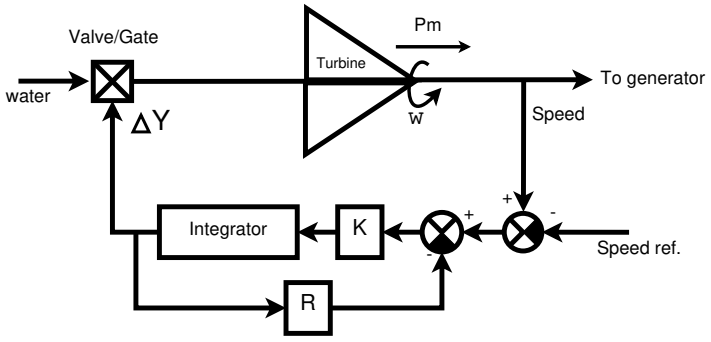


Source: The author

The isochoronous governor is usually employed for regulating a single generator supplying an isolated load, not connected to the public grid. It can also be applied to a multi-generator system in which only one of the generators regulates frequency (the one with the isochronous governor). Two or more generators with isochoronous governors cannot operate in parallel, or else the system would become unstable.

In **Droop Speed Control** mode the governor is not attempting to maintain a constant speed. The main purpose of this mode is to allow two or more generators to operate in parallel, "sharing" the load among them. Figure 12 illustrates the block diagram representation of a generator with droop governor, where Y represents the gate position (control variable) and ω represents the rotor speed.

Figure 12 – Block diagram of generator with droop governor.

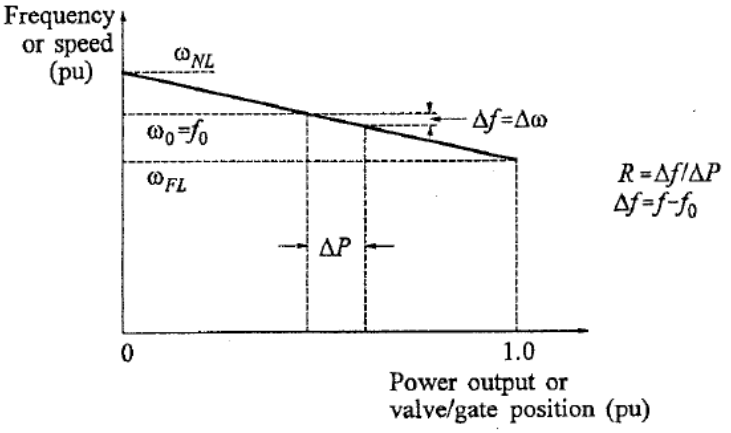


Source: The author

The droop governor has a transfer function characterized as a *proportional controller* with gain $1/R$, where R is referred to as the "speed regulation" or "droop" (KUNDUR *et al.*, 1994). The value of R determines the steady-state speed versus load characteristic of a particular generator. Figure 13 illustrates the ideal response of the droop governor to load variations.

The ratio between speed deviation $\Delta\omega$ (or frequency deviation Δf) to the change in

Figure 13 – Steady-state characteristic of a droop governor.



Source: Kundur *et al.* (1994)

gate position ΔY (or power output ΔP) is equal to R, and therefore can be expressed as:

$$R_{\%} = \frac{frequency\,change_{\%}}{power\,output\,change_{\%}} \times 100\% = (\frac{\omega_{NL} - \omega_{FL}}{\omega_0}) \times 100\% \tag{4}$$

where ω_{NL} represents the steady state speed at no load, ω_{FL} represents the steady state speed at full load and ω_0 is the generator's nominal (or rated) speed. The usual values of droop for real governors are in the range of 2 to 5 percent. It means that, for instance, if a generator governor adjusted to 5% droop senses a frequency deviation of 5%, it will cause a 100% change in power output.

Droop Speed Control, in fact, refers to the fact that the energy being admitted to the prime mover of the synchronous generator is being controlled in response to the difference between a speed (frequency) setpoint and the actual speed (frequency) of the prime mover. To increase the power output of the generator, the operator increases the speed setpoint of the prime mover, but since the speed cannot change (it's fixed by the frequency of the grid to which the generator is connected) the error, or difference, is used to increase the energy being admitted to the prime mover. So, the actual speed is being allowed to "droop" below its setpoint. Any multi-generator power grid usually employs droop speed governors, and therefore, by definition, has non-zero steady state frequency error.

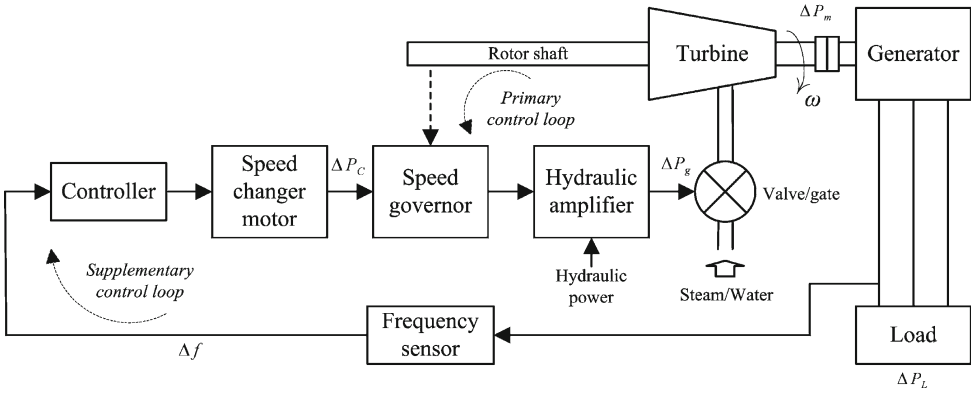
3.2.1.3 Secondary Regulation: Load Frequency Control

In any power system the frequency is dependent on the real power balance between generators and loads (BEVRANI, 2008). If power demand at one point of a network changes, the whole system's frequency is affected. As seen in Section 3.2.1.2, the grid's generators operate

in droop mode, which means variations in load will inevitably produce steady-state frequency deviations. This deviation from nominal values can be viewed as a measure of system generation and load imbalance. In order to correct this imbalance, the set points of at least some governors in the grid must be adjusted. This job is done by the Automatic Load Frequency Controller (ALFC). The process of set point adjustment is called *secondary regulation* or *supplementary control*.

Figure 14 schematically illustrates the relationship between the primary and secondary control loops. By means of each individual speed governor, all generating units contribute to the overall change in generation, irrespective of the location of the load change, using their speed-governing capabilities. This decentralized action is insufficient to bring system frequency back to the rated values, thus the need for the supplementary control loop.

Figure 14 – Schematic block diagram of speed and frequency control



Source: Bevrani (2008)

The supplementary control actions, on the other hand, are performed at a central location and usually employ only a subset of the generators in the system. This supplementary action is considerably slower than the primary, in the range of seconds to minutes (BEVRANI, 2008). The centralized frequency controller is usually part of a larger management system, implemented on top of a computerized infrastructure and communication network known as SCADA/EMS.

3.2.2 Voltage and Reactive Power Regulation of Synchronous Generators

In this section, we briefly discuss the modeling of synchronous generator voltage control and power plant level voltage/reactive power control. Modeling the system-wide problem of voltage control is beyond the scope of this discussion.

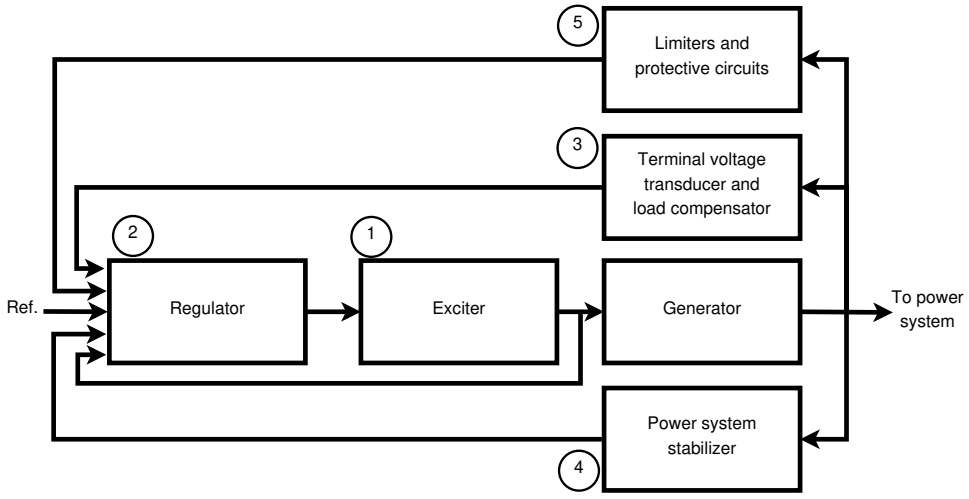
The basic idea of primary and secondary controls also applies to voltage regulation, except that reactive power cannot be efficiently transmitted over long distances (KUNDUR *et*

al., 1994). Synchronous generators are equipped with an Automatic Voltage Regulator (AVR), which continuously regulates armature voltage by controlling the excitation current. Therefore, the synchronous generator can be set to either absorb or supply reactive power, constrained by its minimum/maximum nameplate ratings, like field current, armature current and terminal voltage (KUNDUR *et al.*, 1994). Additionally, at the plant level, a reactive power sharing and joint voltage control scheme is usually present (CALOVIC; JELIC, 1992).

3.2.2.1 Primary Regulation: the Generator's Excitation System Model

The main purpose of the generator's excitation system is to provide Direct Current (DC) to the synchronous machine field winding. Additionally, several control and protective functions are performed with the purpose of fulfilling operational requirements. In this work, we are particularly interested in voltage and reactive power flow control, but other functions include the enhancement of power system's stability and protective actions to ensure that the equipment's capability limits are not exceeded. Figure 15 shows the functional block diagram of a typical large synchronous generator excitation control system.

Figure 15 – Functional block diagram of a large generator excitation control.



Source: The author, based on Kundur *et al.* (1994)

The main components of the system are described below. In this work we are particularly interested in the interaction of the regulator/exciter/generator, therefore components like the PSS and the protective circuits are out of scope.

1. The **exciter**, provides the DC current to the generator's field winding. Works like a "power amplifier" for the signal generated by the voltage regulator.

2. The **regulator** combines, processes, and amplifies various input control signals to the level required by the exciter's input stage. This is the component that receives the voltage setpoint from the secondary voltage control loop.

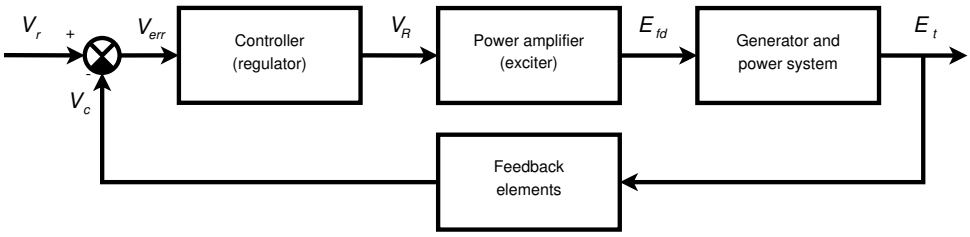
3. **Terminal voltage transducer and load compensator** senses the generator's terminal voltage and compares it to the reference setpoint, which represents the desired output voltage.

4. The **Power System Stabilizer (PSS)** provides the additional input signal to the regulator to damp power system oscillations.

5. **Limiters and protective circuits** include several control and protective functions that ensure equipment's ratings are not exceeded, minimizing the risk of damaging the generator and the excitation system itself.

Mathematical models of the excitation system are essential for performance assessment, planning, and power system operations, as well as for the design and coordination of supplementary control schemes. Figure 16 represents the overall excitation control system in terms of classical feedback control loops. The small signal performance of a typical excitation system can be considered effectively linear (KUNDUR *et al.*, 1994), and such a linear model provides the means to evaluate the closed-loop response of the excitation system to incremental changes in system conditions.

Figure 16 – Classical feedback control model for the excitation system.

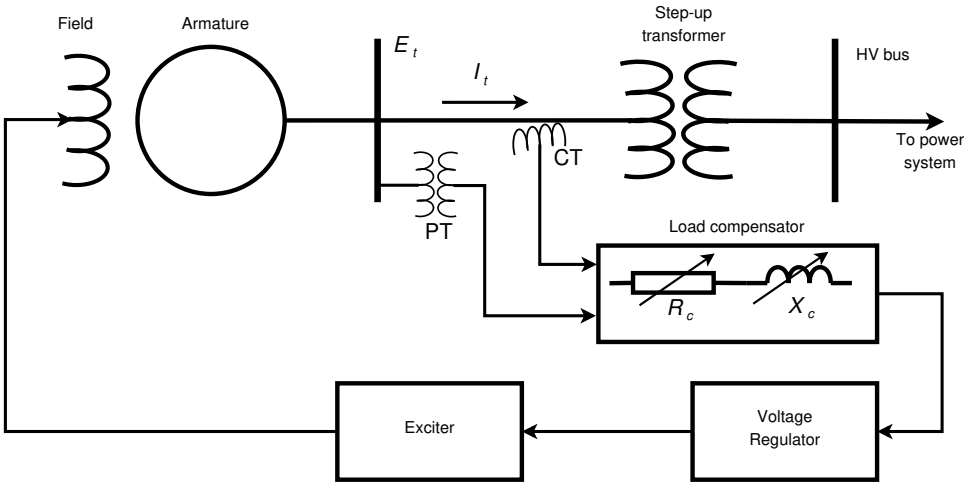


Source: The author, based on (KUNDUR *et al.*, 1994)

The load compensator mentioned above in Section 3.2.2.1 is represented schematically in Figure 17. The voltage drop added to the generator's terminal voltage by the compensator provides a droop effect to the AVR. This droop ensures proper sharing of reactive power between generators connected to the same busbar, either directly or through individual step-up transformers. Without this provision, one of the generators of the set would end up with a slightly

higher terminal voltage than the others and hence would tend to supply all the required reactive power, while the remaining ones would tend to absorb reactive power (KUNDUR *et al.*, 1994).

Figure 17 – Schematic diagram of a load compensator into the voltage control loop.



Source: The author, based on (KUNDUR *et al.*, 1994)

3.2.2.2 Voltage and Reactive Power Reference - Secondary Regulation

In a manner similar to the concept of droop for the speed governor, the AVR's load compensator ensures that generators can operate in parallel satisfactorily, however, it introduces a steady state control error at the power plant's high-voltage busbar.Therefore, supplementary control actions are necessary in order to bring the output voltage back into the acceptable operating range. These supplementary actions, known as secondary voltage regulation, are determined on a system-wide basis by the regional load dispatcher or system operator authority. Large power plants participate in the secondary voltage regulation by following a voltage or reactive power reference determined by the regional dispatcher. This reference can be relayed either automatically through digital communication infrastructure, scheduled along with the accorded interchange schedule, or directly communicated, on demand, by the regional dispatcher to the power plant operator.

At the power plant level, the function responsible for following this reference is known as *Joint Bus Voltage/Reactive Control (JBVRC)*, sometimes also referred to as Joint Voltage Control (JVC), or simply Automatic Voltage Control (AVC). In modern power plants, the JBVRC is also implemented on top of the SCADA/EMS and includes the features of common plant bus voltage control and distribution of reactive generation to individual units in operation (CALOVIC; JELIC, 1992).

3.3 SOFTWARE COMPONENTS AND POWER SYSTEMS CONTROL

As discussed above, many types of controls are integrated into the power system, including generator excitation controls, prime mover controls, generation or load tripping or shedding, fast fault clearing (protection), high-speed re-closing, reactive power compensation, load–frequency control, and other special controls (BEVRANI, 2008). These controls can be organized hierarchically by levels, and classified according to operation modes, conditions, and time constants of the control loops, as shown in the following sections.

3.3.1 Hierarchical Organization of Power Plant Controls

Power plant controls are organized into a hierarchical structure classified by the IEEE 1249-2013 Standard (IEC/IEEE, 2013). According to the *location*, IEEE 1249-2013 classifies power plant controls into three levels: *Local, Centralized* and *Off-site*.

1. The lower level, or *Local Controls* are co-located with the controlled equipment, or placed within sight of the equipment. Examples of such controls include each of the generating unit's speed governors and AVR.

2. An intermediate level, or *Centralized Control* is located at a remote location from the point of view of the controlled equipment, but within the limits of the power plant itself. This level usually concentrates all (or at least most of) the necessary controls for operating all the plant's equipment from a single location, under normal conditions. Examples of these controls are the joint control of active power, switchyard control, and spillway control.

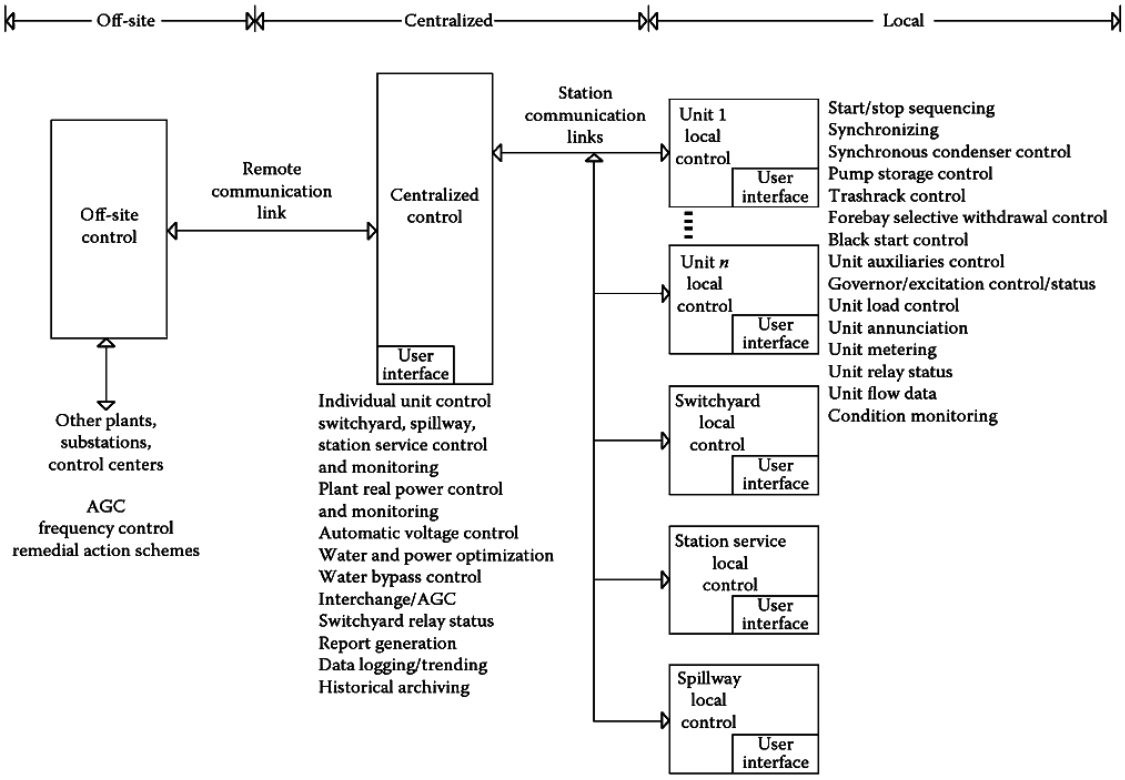
3. The higher level is the *Off-site control*, which constitutes an operations control center capable of remotely controlling several power plants. Each of the individual power plants may or may not be attended by an in-site operations staff.

A diagram depicting the relationships between these control levels is shown in Figure 18.

By observing Figure 18, it becomes clear that such a control hierarchy depends on efficient control methodologies, communication infrastructure, and information technology (IT) services. In contemporary control technology, all the power plant control levels include software components. Therefore they fit the definition of *Cyber-Physical System* (LEE; SESHIA, 2011)[6].

[6] In comparison, Strasser *et al.* (2020) used the term *Cyber-Physical Energy System* to refer to the concept of "Smart Grid".

Figure 18 – Relationship between local, centralized and off-site control.



Source: IEEE Std. 1249 - Guide for Computer-Based Control of Hydroelectric Power Plant Automation – (IEC/IEEE, 2013).

3.3.2 Classification of Power Plant Controls

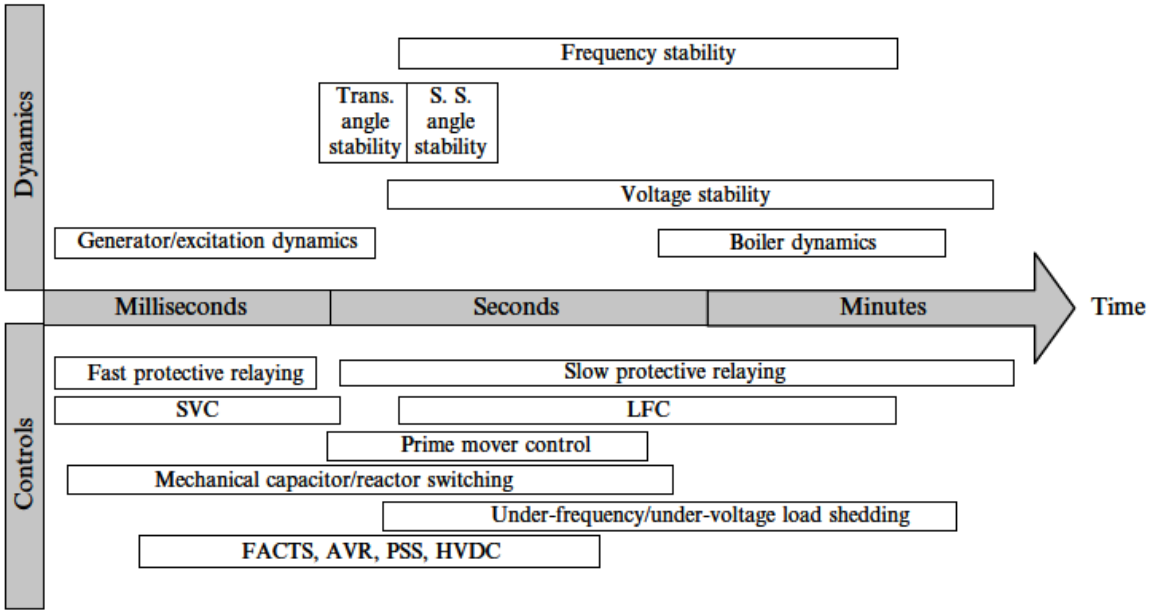
With respect to the **operation mode**, power system controls can be classified into two major categories (BEVRANI, 2008): (1) continuous and (2) discontinuous controls. *Continuous controls* operate directly on plant equipment, including generator excitation controls (PSS and AVR), prime mover controls (speed governors), reactive power controls, and AGC. They are usually linear, continuously active, and use local measurements. The *discontinuous controls* are meant to stabilize the system after disturbances or once off-nominal conditions are detected, therefore they operate only after certain triggering events occur in the power system. They perform actions such as generator/load tripping, capacitor/reactor switching, and other protection plans. These power system controls may be local at power plants and substations, or over a wide area. Examples of discontinuous controls include discrete supplementary controls, Remedial Action Schemes (RASs), and emergency control/protection schemes (NEIS *et al.*, 2012; NEIS *et al.*, 2012; NEIS *et al.*, 2022).

With respect to the **operating condition**, power system controls can again be divided into two categories: (a) normal/preventive controls; and (b) emergency controls. *Normal/preven-*

tive controls are applied while the power system operates under normal and alert states[7], to stay in or return into normal conditions. *Emergency controls* are applied in an emergency or *in extremis* state, in order to stop the further progress of a failure and return the system to a normal or alert state. Automatic frequency and voltage controls are part of the normal and preventive controls, while some of the other control schemes such as under-frequency load shedding, under-voltage load shedding, and special system protection plans can be considered under emergency controls.

With respect to the time constants of the control actions, it is hard to establish a crisp classification of power system controls. However, it is widely accepted that control loops at lower system levels, (e.g. at the generator level) are characterized by smaller time constants than the control loops active at a higher system level, like the centralized and off-site. For example, the AVR, which regulates the voltage of the generator terminals to the reference value, responds typically in a timescale of a second or less. On the other hand, controls like the Load Frequency Control (LFC) and the secondary voltage regulation loop, operate in a timescale of several seconds or even minutes (BEVRANI, 2008). Figure 19 schematically represents different timescales for some of the power system controls, along with the associated dynamic phenomena[8].

Figure 19 – Different timescales of power system dynamics and controls.



Source: Bevrani (2008).

[7] Power system operating states can be conceptually classified into five states: normal, alert, emergency, *in extremis* and restorative(KUNDUR *et al.*, 1994).

[8] Note that the term SVC in this figure refers to *Static Var Compensator*, not to be confused with the concept of *Secondary Voltage Control*.

3.3.3 Energy Management and Automation

In power systems operation, **energy management** is defined as *"the process of monitoring, coordinating, and controlling the generation, transmission, and distribution of electrical energy"* (GRIGSBY, 2007). The physical system being managed includes power plants, transmission networks (grid), and load centers. Current technology for electrical transmission and distribution systems cannot provide significant energy storage, therefore supply and demand must be balanced by controlling either generation or load (or both). Power production is controlled by turbine governors and voltage regulators at the generator level, and by automatic generation, voltage, and reactive power control functions performed at centralized and/or off-site levels. These centralized control functions are part of a suite of software applications known as the EMS.

EMS functions at the centralized and off-site levels are dependent on a data acquisition and telecontrol infrastructure responsible for collecting status and measurement information needed to supervise overall operations, as well as issuing control signals like switchgear operation and setpoint adjustments. This infrastructure is known as the SCADA layer. A SCADA system usually is composed of a master station communicating with several, distributed, Remote Terminal Units (RTUs). RTUs collect and upload process data to the SCADA master station, allowing operators to observe and control physical plants.

In this work, the centralized and off-site software control functions running on top of a SCADA layer will be generically designated as "**energy management**" or EMS applications. On the other hand, local control functions, generally running on Programmable Logic Controller (PLC) platforms or Intelligent Electronic Devices (IED)[9], will be generically designated as **"automation functions"**.

3.3.4 Development Process of Energy Management Applications

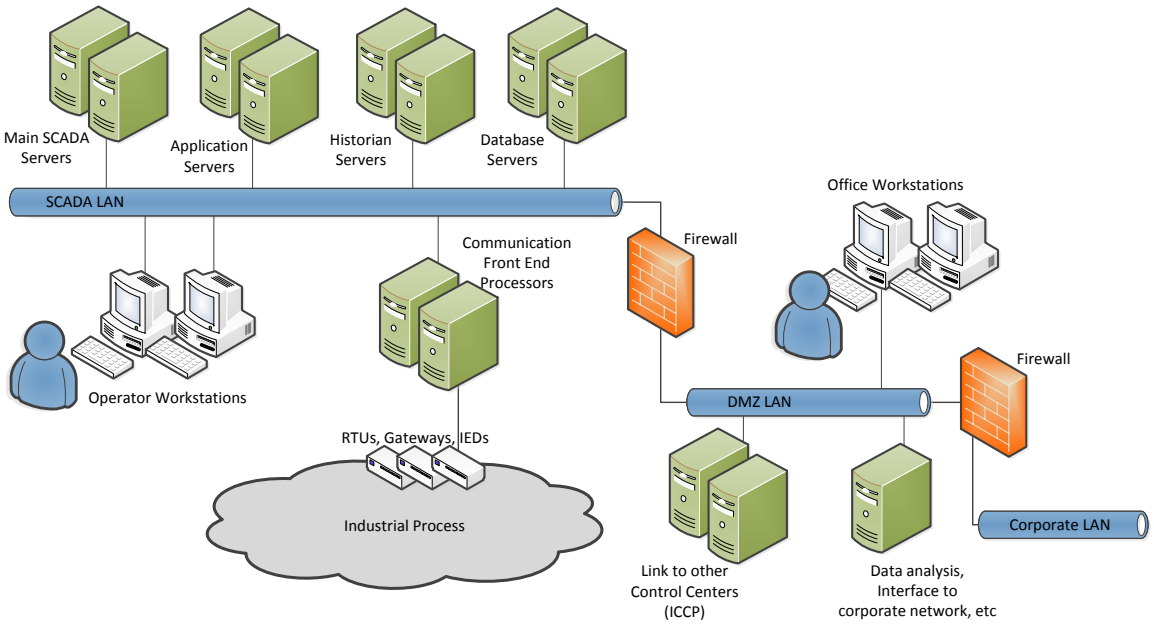
In a broad sense, hydropower plant control and other power system-related SCADA/EMS projects are usually described in the specialized literature at high-level overview, such as in references (COHEN *et al.*, 1989; SKOPP; VARADAN, 2000; AZEVEDO; OLIVEIRA-FILHO, 2001; CORERA *et al.*, 2005; VIRMANI; SAVULESCU, 2008; BJÖRKMAN *et al.*, 2010; ABB Inc., 2016; AGARWAL *et al.*, 2016), giving very few details about the software

[9] IED is the term used to describe microprocessor-based controllers of power system equipment.

process involved in its development. Despite that, common traits can be identified in the design of the so-called “open architecture” SCADA/EMS systems.

A typical simplified SCADA/EMS architecture is represented in Figure 20[10]. In this figure, we schematically show how the physical industrial process interfaces to its cyber counterpart, the SCADA/EMS: through field devices such as RTUs, IEDs, and gateways to other data acquisition systems. The front-end communication processors collect data from the field devices, perform pre-processing and deliver it to the core SCADA real-time database; they also communicate supervisory commands back to the field. The main SCADA servers are the heart of the system, supporting the bulk of SCADA functions, maintaining the real-time database, and providing data to the operator workstations. The operator workstations connect directly to the core SCADA servers and are the entry point to process supervision and human-triggered control actions. The application servers run the EMS applications, typically having direct access to the SCADA real-time database and being able to relay automatic control actions to the field. The historian servers collect real-time data and store it in a time-series database. Relational database servers are the main repository for system configuration parameters, such as RTU communication configuration, global system options, and network models.

Figure 20 – A typical, simplified SCADA system architecture.



Source: The author.

[10] Depending on the size of the installation, this configuration can be scaled either up or down, for instance: smaller installation may incorporate application servers into the main SCADA servers, and optionally may not have the historian functionality; larger installations may incorporate other functionalities like engineering workstations, training, and development systems.

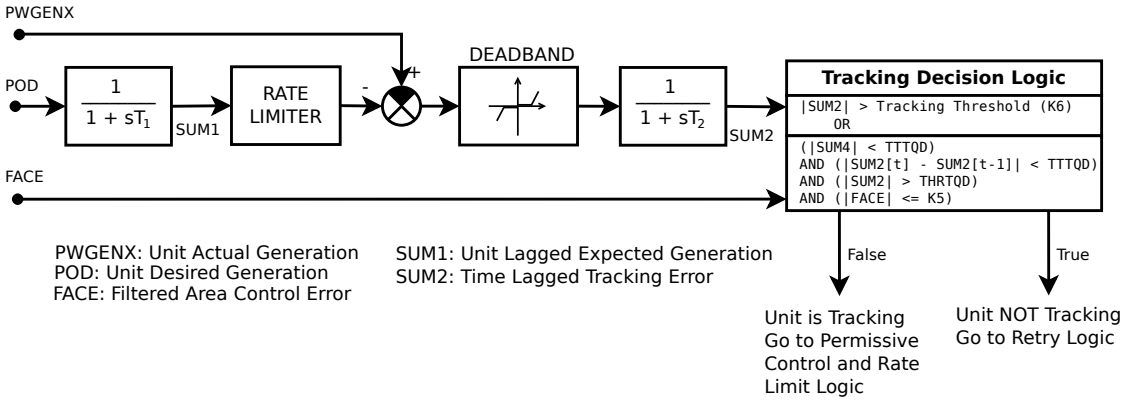
Energy management applications are typically deployed into the application servers previously represented in Figure 20. They are usually specified and designed using modeling paradigms borrowed from control theory. Such paradigms include block diagrams and transfer functions in the continuous "s" or discrete "z" forms. A vast number of publications can be found in the specialized literature demonstrating the use of such paradigms for modeling EMS applications (NEIS *et al.*, 2019). Those researches are generally focused on the theoretical aspects of the modeling, such as in references (ROBERT; HURTADO, 2008; SUN *et al.*, 2009; SULLIGOI *et al.*, 2011; BEVRANI *et al.*, 2012; BEVRANI; HIYAMA, 2009; MARTÍNEZ *et al.*, 2012; SHENG *et al.*, 2009; BAEK *et al.*, 2013; BAEK, 2014; CORSI *et al.*, 2004a; CORSI *et al.*, 2004b; HARVEY *et al.*, 2017; AZIZI; KHAJEHODDIN, 2018; LOU *et al.*, 2019) which offer little or no information at all about the software engineering process, i.e.: the process of transforming the specified models into software artifacts.

However, based on our experience working with both the power utilities and the EMS software providers, it is possible to state that in the EMS area, models are used mostly as sketches for the function specification or documentation like tuning guides, maintenance, and user manuals. Even when used as blueprints of the designed system, models tend to quickly become obsolete, since updates performed in the application's 3GL source code are not always reflected in the system models.

Let's take as an example the model depicted in Figure 21. This model represents the desired behavior of the tracking detection logic for Itaipu's AGC. This logic is responsible for detecting the occurrence of malfunctions that prevent a generating unit's governor from properly following its LFC reference. Such malfunctions may occur due to local or remote defects in either hardware or software and will usually manifest as one of the following symptoms: (1) the unit's measured power drifts away, while its governor fails to respond to corrective control signals; (2) the unit's governor fails to respond to control commands.

The tracking detection logic is specified in a block-diagram-like notation, containing both continuous "s-form" transfer functions, logical and arithmetical operators. At some point, this model was manually translated by a developer into a 3GL programming language and originated a subroutine module that was compiled and built into the executable AGC. This manual translation process also includes the transformation of the continuous time transfer functions into discrete time, since the SCADA/EMS software is completely based on digital computers and operates over (virtually) uniformly sampled signals. Additionally, we need to

Figure 21 – A model of the unit tracking detection logic in Itaipu's AGC.



Source: The author, based on documents from Itaipu.

consider that the model in Figure 21 is an abstraction of the "thing being modeled", and therefore may not include all the necessary details that the developer adds into the 3GL module in order to implement the actual tracking detection logic. The manual translation of the model presents several risks and challenges, for instance:

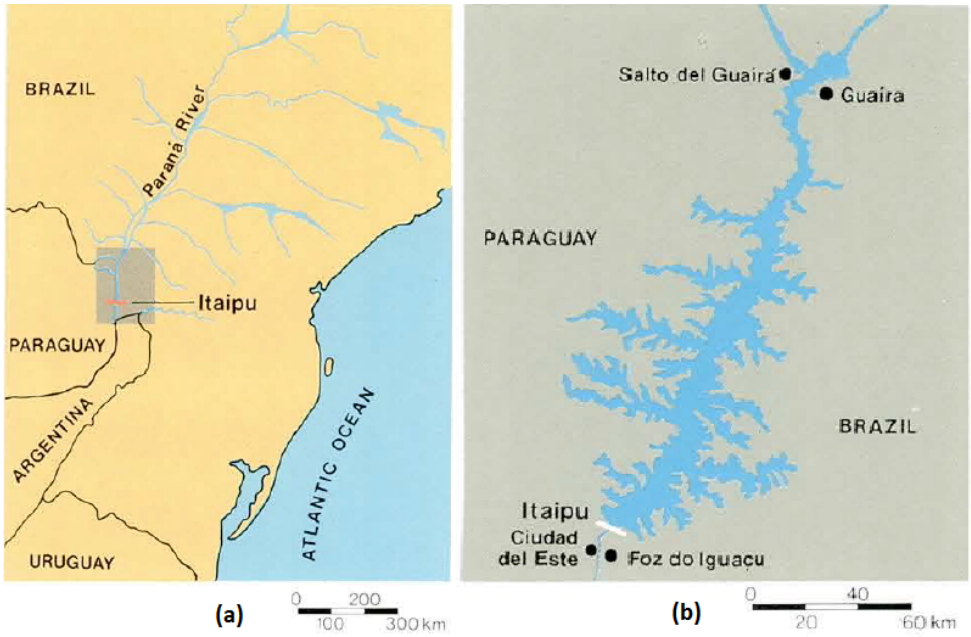
- it may introduce errors or deviations from the designer's intended behavior;
- future updates to the design have to be performed both in the model and in the source code;
- chances are that modifications will end up being made to the source code alone, while the model may remain outdated;
- similar models may end up being expressed as very different implementations in 3GL, particularly when the coding is performed by different developers;

In this scenario, we propose to adopt an MDE approach to the software engineering process, which is expected to help cope with these and other challenges, as we will further discuss in Chapter 5.

3.4 THE ITAIPU CASE STUDY

The Itaipu Hydroelectric Project is a bi-national project undertaken by two neighboring countries, Brazil and Paraguay. It is located on the Paraná river, which delimits the border between the two countries, 14 km upstream from the international bridge joining the cities of *Foz do Iguaçu* in Brazil and *Ciudad del Este* in Paraguay, as illustrated in Figure 22 (COTRIM, 1994).

Figure 22 – Itaipu project location map.



Source: Adapted from (COTRIM, 1994).

3.4.1 Overview of Itaipu Project

Since 2007 Itaipu power plant is composed of 20 generating units[11], each one of them rated at 700 MW nominal power. Therefore the total installed capacity is 14000 MW.

Given the difference in frequencies used in each country – 60 Hz in Brazil and 50Hz in Paraguay – half of the units operate at 50 Hz and half operate at 60 Hz. The power demand from the Paraguayan system is lower than the total 50 Hz installed capacity, particularly during the early operating stages of the project (COTRIM, 1994). The surplus power generated at 50 Hz is interchanged with the Brazilian system, by means of static conversion to High Voltage Direct Current (HVDC) and transmission at ± 600 kV to the São Paulo area, in Brazil.

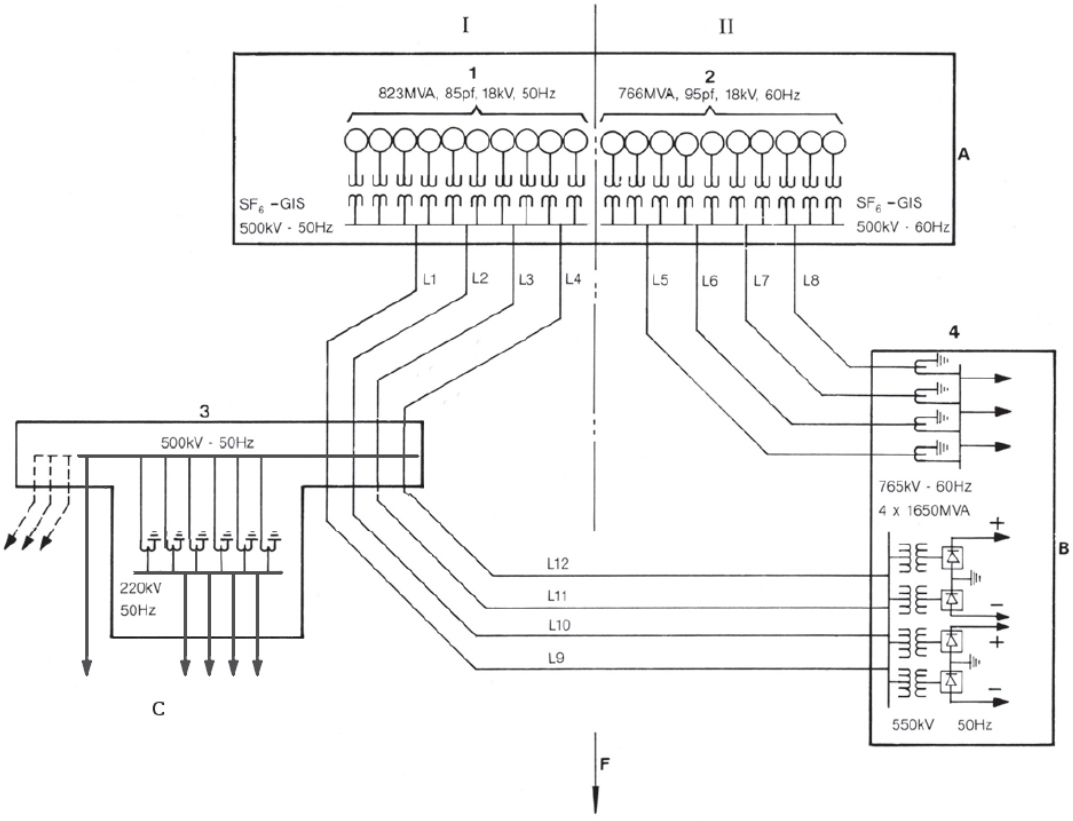
Itaipu plant can be viewed as the equivalent of two powerhouses: one of them operating at 60 Hz frequency and having 7000 MW installed capacity, synchronized to the Brazilian system; the other, with equivalent installed capacity and operating at 50 Hz, supplying power to the Paraguayan system and the DC link with Brazilian system. The powerhouse, substations, and associated transmission system are roughly composed of:

- 20 generating units, half of them operating at each country's nominal frequency, and rated voltage at 18 kV;

[11] The original project was composed of 18 generating units, and two additional units were later commissioned.

- Transformer banks associated with each unit, stepping up the voltage to 500 kV;
- 500 kV gas insulated switchgear and substation (GIS-SF_6)[12] installed inside the powerhouse;
- 500 kV transmission, using four lines for each frequency sector, interconnecting to both right and left bank areas;
- 750 kV AC and ± 600 kV DC for transmission into Brazil;
- 230 kV AC and 500 kV AC for transmission into Paraguay.

Figure 23 – Itaipu simplified electrical diagram.



Source: Adapted from (COTRIM, 1994).

Figure 23 illustrates a simplified electrical diagram of the Itaipu project and its interconnections. A central line schematically represents the border between Paraguay (I) and Brazil (II), and the arrow (F) represents the Paraná river flow direction. The Itaipu plant (A), containing both the 50 Hz (1) and 60 Hz (2) units, connects to the Brazilian (B) and Paraguayan (C) systems.

[12] Gas Insulated Switchgear (GIS) employs sulfur hexaflouride (SF_6) as the insulator, an inert gas having two to three times the insulating ability of air at the same pressure.

This connection is made through the right river bank substation (3) in 50 Hz, and the Foz do Iguaçu substation[13] (4) in 50 and 60 Hz, both by means of 500 kV lines. The right river bank substation (3) is also interconnected to the 50 Hz sector of Foz do Iguaçu substation (4), where the static HVDC converters are located.

3.4.2 Itaipu Power Plant Control and Supervision

The power plant control of the generating units and ancillary equipment can be performed from two different locations: the unit local control rooms and the power plant Central Control Room (CCR) – respectively the local and centralized levels, according to IEEE 1249-2013 Standard (IEC/IEEE, 2013). The original control system installation employed hardwired circuitry to perform power plant control functions (COTRIM, 1994). Currently, most of the control and supervision functions are performed by means of a digital supervisory control and data acquisition system (SCADA). The hardwired system serves mostly as a backup to the digital system.

The overall features of the Itaipu power plant control system are:

- A central control room located in the powerhouse main structure: the CCR has facilities for control and supervision of the power plant, spillway, and the right bank substation.
- Local Control Rooms (LCRs): There are ten local control rooms in the powerhouse, one for each pair of generating units.
- Right bank substation control room, located at the right bank substation: It has facilities for controlling the transformers/regulators and 500/220 kV transmission lines which deliver 50 Hz power to Paraguay and to the Foz do Iguaçu HVDC converter station in Brazil.
- Spillway local control room, located at the left side of the spillway structure which has controls for opening and closing the fourteen spillway gates.
- Load Dispatch Center (LDC)[14]. The main functions of the LDC are to forecast the availability of Itaipu generation and to correlate it with the needs of the two power systems.

[13] Foz do Iguaçu substation does not belong to Itaipu, and is operated by Furnas company.

[14] Because of the project importance and the role that Itaipu production plays in the operation of Brazilian and Paraguayan power systems, the complexity of hydrological conditions and numerous electrical transmission interconnections with the power utilities, the power plant has an LDC (COTRIM, 1994).

The normal operating mode for the power plant is centralized. Operation from the LCR is primarily used during commissioning, testing, initial operation, and in case of emergency due to contingencies in the CCR. In general, all control levels are mutually exclusive: each device is either under “central” or “local” control, but never simultaneously under both. Exceptions to this rule include certain equipment functions that must operate independently of the chosen control level, for example: protective action, such as circuit breaker trips; unit emergency shut-down commands; lowering of intake gates; instrumentation and alarm annunciation.

In this work, regarding the Itaipu digital control system, the term “centralized control action” refers to both: (a) human-triggered control actions initiated from the CCR or LDC; (b) automatic control actions initiated from EMS functions on the digital control system.

The digital control system, regarded as “SCADA/EMS”, provides a modern and highly flexible platform for control and supervision of the complete power plant, and the performing of energy management functions (COHEN *et al.*, 1989). The digital control system is basically composed of:

- Remote terminal units, distributed throughout the power plant, usually close to the equipment or process being supervised and controlled. The RTUs are capable of collecting data from transducers and other field equipment and performing actions on actuators according to commands received through the communication link.

- Operating stations, also referred to as “consoles”, distributed among the CCR and LDC. The consoles are the interface between the operations personnel and the digital control system.

- A central computerized station, responsible for communication and data processing. The central station also hosts the execution of a set of automatic functions, regarded as “EMS” functions (see section 3.3.3).

The central station functions can be further grouped and described as follows:

- General functions like data acquisition from RTU, data processing and recording, routing information to displays on operator consoles, processing of operator actions, issuing commands back to the RTU, and exchange of information with other internal and external systems.

- Continuous or discontinuous control functions like automatic generation control, automatic voltage control, and certain special protective actions.
- Operations assisting functions such as hydrological forecasting, generation and maintenance scheduling, and provision of electrical and hydraulic system operation guidelines.
- Equipment and network security analysis functions, like state estimation, load flow, unit operating point, and capability monitoring.
- Post-operation reports like the elaboration of statistical data, operational reports, and production accounting.
- Other miscellaneous functions such as software database management, display development and maintenance, dispatcher/operator training, and development environment.

Itaipu's digital control system was deployed in the early 2000s and has already undergone two upgrades. It was supplied by a well-known vendor of power system energy management and automation systems, and includes a considerable degree of customization and special functions developed exclusively for the project. A certain degree of customization was deemed necessary given the particular control requirements dictated by the characteristics and complexity of the Itaipu project. However, customized and unique functions add extra complexity, cost, and risks to the project, particularly in the event of upgrades or possible migration to a different product line. Therefore, one of the goals of this work is to study and apply state-of-the-art software engineering techniques that can help cope with the complexity, mitigate risks and possibly reduce costs of the life cycle management of a centralized digital control system. The Itaipu digital control system is a practical case study to which we intend to apply our proposed methodology.

3.4.3 Automatic Control Functions in Itaipu

The automatic centralized control functions currently in service at the Itaipu power plant are responsible for implementing, among other things: supplementary control actions, joint generating unit control, and emergency control schemes, as well as, in certain conditions, operating under off-site (remote) control mode. Some of these functions are of particular interest for the development of this research and thus are described in the following sections.

3.4.3.1 Automatic Generation Control

The Automatic Generation Control (AGC) consists of a set of functions operating online in real-time, to adjust the generation against the load at minimum cost (GRIGSBY, 2007). Some of the functions that can be classified under the AGC umbrella include:

- LFC: this is the function responsible for balancing the total generation and load (plus losses) in order to maintain the nominal system frequency (BEVRANI, 2008);
- Joint Control of Active Power (JCAP), also known as "load sharing": provides a plant with the ability to receive a single MW generation setpoint that may be shared among a group of several turbine/generators (OLIVEIRA *et al.*, 2017). The plant MW setpoint may be either entered at the plant itself or received from an off-site control center (System Operator, for example);
- Area Interchange Control and Scheduling: is the function responsible for maintaining net power interchanges with neighboring control areas at the scheduled values (GRIGSBY, 2007);
- Economic Dispatch: maintains power allocation among units at economically desired values;
- Spinning reserve monitoring: monitors the amount of spinning reserve maintained on the system, and alerts the operators in case of violations. Maintaining enough reserve capacity is required in case generation is lost (GRIGSBY, 2007);

With respect to the operation mode, AGC is a continuous control. With respect to the operating condition, AGC can be classified as normal/preventive control.

Itaipu performs or takes part in the performance of these AGC functions, which are implemented on top of the centralized digital control system[15]. Several control strategies are available in Itaipu's AGC, but the most commonly adopted, according to each powerhouse's frequency sectors, are:

[15] A hardwired power plant control function called *Joint Turbine Control (JTC)* is still operational, serving as a backup to the digital AGC function.

- **50 Hz sector** performs LFC supplementary control actions, at Constant Frequency control mode (CF)[16];
- **60 Hz sector** can use different strategies (depending on the situation): (a) load sharing with remote setpoint control from System Operator; (b) load sharing in an open (secondary) loop;

3.4.3.2 Plant Level Joint Bus Voltage and Reactive Power Control

Current Itaipu's digital JBVRC function is called Automatic Voltage Control (AVC)[17]. It is a real-time function responsible for maintaining the required voltage level at designated generator-controlled bus bars while performing an optimum distribution of reactive power among the participating generators. AVC's target control variable is the voltage level at the 500 kV bus bar in the powerhouse. That voltage level is controlled by changing the generators' terminal voltage setpoints and thereby, their reactive power output.

Different control strategies, also called modes, are also available for AVC, for example:

- Closed loop reactive power sharing among generators;
- Closed loop excitation current sharing among generators;
- Open loop reactive power sharing.

AVC function currently operates under the same mode in both powerhouse sectors, although different parameter tuning is used. Just like AGC, AVC can be classified as a continuous operation mode function, and as normal/preventive control.

3.4.3.3 Centralized Emergency Control Schemes

A special emergency control scheme is implemented on top of the digital control system, which monitors the condition of certain equipment and takes remedial control actions in case the operating point of this equipment exceeds the rated capacity. The goal of this scheme is to automatically reduce the 60 Hz power production once any of the 500/765 kV transformers in

[16] With future energy integration with the Yacyretá Power Plant, the operation under Constant Net Interchange (CNI) mode is being evaluated.

[17] A hardwired power plant control function called JVC is still operational, serving as a backup to the digital AVC function.

the associated transmission system becomes overloaded by more than 10% of its rated capacity (1800 A or 1650 MVA) (NEIS *et al.*, 2012; NEIS *et al.*, 2012). Once such overload has been detected, the scheme calculates a new target power production for the 60 Hz sector, such that all the transformers are brought back below rated capacity, and trigger the corresponding AGC generation reschedule.

This special scheme can be classified as a discontinuous operation mode function, that operates under emergency conditions.

3.4.4 Simulator and Production Environments

SCADA/EMS products are usually shipped with a Dispatcher Training Simulator (DTS) package. The DTS is a realistic simulation environment capable of representing the dynamic response of the power system under a wide range of operating conditions, within the limitations of the power system model (WANG *et al.*, 1994). DTS employs the same set of displays, software tools, and databases that compose the production SCADA/EMS used in the power plant operations, except for the field process data acquisition, which is simulated. The simulation is based on load flow solutions and dynamic models of power system elements such as generators and loads. Such dynamic models satisfactorily represent transient electromechanical effects that are active for a period of one second or longer, showing pronounced effects on the power system magnitudes.

Besides training, the DTS has proved to be a useful tool for after-the-fact power system disturbance investigation, testing database changes, and validating EMS software changes. The DTS has been used for years to verify the performance of applications prior to the deployment and test on the power plant's SCADA (HAQ *et al.*, 2009). Since, as explained above, the DTS runs the same software as the production system, virtually any application that correctly executes within DTS can be safely deployed to the production system.

3.5 CHAPTER SUMMARY

This chapter summarizes some of the paradigms specific to the power system's domain, upon which the remainder of this work is based. An overview of the hydropower production process is presented, along with a description of some of the software components associated with the process control, thus configuring a Cyber-Physical System. A particular case of interest,

the Itaipu Power Plant, with its basic equipment and controls configuration, is briefly described. The motivation and contextualization for this research are also briefly presented: the application of state-of-the-art software engineering techniques to cope with the complexity, mitigate risks, and possibly reduce costs of the life cycle management of a centralized digital control system. Results of this research, although based on one particular case study, can certainly be extended to other scenarios and thus bring contributions to the energy management and software engineering fields.

We have also argued that the paradigms used by domain specialists for specifying requirements and describing intended behavior in the power systems area usually involve notations such as block diagrams and transfer functions. However current software modeling tools lack support for these paradigms. Not surprisingly, our literature review has shown that MDE approaches currently play a rather limited role in the development of EMS applications. Some questions regarding this matter remain open until now, for instance: (i) Which domain-specific languages can be used for modeling EMS applications? (ii) Which transformation techniques and tools can be used? (iii) How can we easily integrate custom EMS applications with commercial SCADA packages? Those are questions we address in the following chapters.

4 LITERATURE REVIEW AND STATE OF THE ART

This chapter presents a literature review and state-of-the-art analysis of software development for power systems and related applications. The findings reported in a systematic literature review on the subject, which we have performed and published in 2019 (NEIS *et al.*, 2019), are discussed and augmented with complementary bibliography published after the original review was performed. Answers to the research questions are outlined, and open problems are identified. Comparatively, the D-SPADES approach is placed in perspective and the gaps it proposes to fill are highlighted.

4.1 REVIEW OUTLINE

The objective of the review was to gather information about the software process applied to the development of power systems applications. With this information, we can identify some of the challenges faced by engineers from the power system area in describing system requirements and transforming abstract models into lower-level software artifacts. We wish to elucidate how domain knowledge is currently being captured from the power system specialist, and how that knowledge is used in high-level system specification, validation, and implementation.

The original review included works from the year 2006 until March/2019. In that review, 40 publications were included. A supplementary review, following the original protocol and using the same digital libraries (*IEEE Xplore*[1] Digital Library (DL) and the *ACM* DL[2]) was performed on May/2022. This supplementary review has identified additional 11 publications considered relevant and therefore included in this chapter. Further details about the review protocol can be found in (NEIS *et al.*, 2019). In addition to the publications found through this review protocol, this chapter also references and discusses other relevant works in the MDE area.

4.1.1 Research Questions for the Review

We have formulated the following research questions that shall be specifically answered in the scope of the literature review:

1. **RQ1:** *What are the overall characteristics of the software process being applied to the*

[1] http://ieeexplore.ieee.org/
[2] https://dl.acm.org/

development of power systems applications? By answering this question we seek to identify the type or format of models representing requirements and specifications (the source model) and the format of the software artifacts into which the source model is transformed. We also wish to identify the transformation process, whether it is automated or manual.

2. **RQ2:** *Is the concept of MDE being applied to the development of SCADA/EMS, or other power system applications?* Answers to this question shall also help identify promising DSLs for this type of application, transformation techniques and tools, and related standards applicable to the integration with commercial SCADA packages.

4.2 INFORMATION COLLECTED FROM THE REVIEWED WORKS

We have categorized the reviewed publications and emphasized some of the most relevant features observed in each category, as shown in Table 2, and discussed in the following subsections. Desirable features observed in works from each category are highlighted, and related to the proposed D-SPADES approach.

Table 2 – Comparison of development approaches from reviewed works.

Sec.	Approach/Work	Application Domain	Source Model	Transformation	Target Model
4.2.1.1	Breesse	Avionics	**Actor-Oriented Simulink/Stateflow**	Automatic M2M	EMF/Ecore
4.2.1.2	AMoDE-RT	Embedded	UML/SysML	**Automatic (M2T)**	**3GL (Java)**
4.2.2.1	IEC Standards and Ontologies	Substation Automation	IEC 61850, ontological model	Automatic	IEC 61499
4.2.2.2	CIM-based	**SCADA/EMS**	IEC 61970 (CIM)	Semi-automatic	**OPC-UA address space**
4.2.3.1	Rapid prototyping for smart grid	Smart Grids	PSAL EMSOnto	**Automatic M2M & M2T**	IEC 61499, 61131, others
4.2.3.2	FMDE4SGRID	Smart Grids	SGAM-based	**Automatic M2M & M2T**	**3GL IDL, Java**
4.2.3.3	ThingML+	Smart Grids Embedded, IoT	DA/ML	**Automatic (M2T)**	**3GL (Python)**
4.2.3.4	Power-Attack	Protection System Cyber security	Power-Attack DSL	Interpreted Language	Discrete Events
4.2.4.1	MPC-based	Hydro unit speed governor	MPC discrete control law	Unspecified	PLC program
4.2.4.2	Event-triggered DG control	**Micro Grids (Volt/Frequency control)**	Discrete control law	Unspecified	Unspecified
4.2.4.3	DFR Algorithm	**SCADA/EMS** and **Micro Grids**	**Algorithm and Transfer Functions**	Unspecified	Unspecified
4.2.4.4	Classical Control Theory	**SCADA/EMS (Volt/Frequency control)**	**Block diagrams & Transfer Functions**	Unspecified	Unspecified
4.2.4.5	NewSART project	**SCADA/EMS (Volt/Frequency control)**	**Block diagrams & Scilab/Scicos model**	Automatic (code generator)	**3GL (C)**
5	**D-SPADES**	**SCADA/EMS**	**Actor-Oriented EMSML/MoML**	**Automatic (M2T)**	**3GL (C++, Fortran)**

4.2.1 Applications not Related to Power Systems

4.2.1.1 Breesse

Breesse is described as a bridge for the EMF ecosystem and the MathWorks Simulink and Stateflow ecosystem (PAZ; BOUSSAIDI, 2020), built-in response to the needs of the avionics systems industry. According to the authors, such systems are represented using a mix of UML with Simulink and Stateflow design models. Manually ensuring consistency between such heterogeneous design models is a resource-consuming and error-prone activity. Breesse is implemented on top of EMF technologies and the Matlab Engine API for Java, being able to import the contents of Simulink and Stateflow design models and libraries into EMF-based representations. It, therefore, provides Eclipse MDE users and tool developers with access to Simulink and Stateflow models and libraries in the form of EMF models, being also able to directly connect to a running Matlab instance. Breesse is part of the toolchain that supports engineering teams in identifying inconsistencies between design models. To flag inconsistencies between Simulink, Stateflow, and UML design models, Simulink and Stateflow design models are imported into Eclipse using Breesse. Then, the imported models are further processed with EMF-based technologies that are part of the toolchain. Therefore, Breesse is not used to generate code, but rather to ensure consistency among different representations of the system, eg: UML and Simulink/Stateflow. Either way, this work is still relevant when compared to D-SPADES, since the source models in the toolchain are AO-based. The work also demonstrates that it is possible to build a EMF-compatible metamodel for Simulink/Stateflow models, which in turn could be used as an alternative representation in D-SPADES.

4.2.1.2 AMoDE-RT

The **Aspect-oriented Model-Driven Engineering for Real-Time Systems (AMoDE-RT)** (WEHRMEISTER *et al.*, 2014; WEHRMEISTER *et al.*, 2013; WEHRMEISTER, 2009; WEHRMEISTER *et al.*, 2007a; WEHRMEISTER *et al.*, 2007b) is an MDE approach tailored for embedded and real-time systems. It combines the use of the UML (MARTE[3] profile) along with concepts of the Aspect-Oriented Software Development (AOSD) to deal with the system's

[3] Modeling and Analysis of Real-Time and Embedded systems (MARTE) is an Object Management Group (OMG) specification of a UML profile adds capabilities to UML for model-driven development of Real-Time and Embedded Systems. See https://www.omg.org/spec/MARTE/1.2/.

crosscutting concerns. AMoDE-RT is supported by a tool chain composed of: (a) an automated and configurable code generation tool, capable of transforming UML models into source code for (virtually) any given target execution platform; (b) an automated testing tool that can execute a set of test cases on the UML model.

In AMoDE-RT, code generation from the UML model is performed through a script-based approach, in which small scripts define how to map model elements into target platform constructions, generating source code fragments that are merged to produce source code files. D-SPADES, on the other hand, employs a specialized Model-to-Text (M2T) transformation language for code generation, besides being targeted specifically at SCADA/EMS applications and thus using a specialized Domain Specific Language (DSL) to express models, as emphasized in Table 2.

4.2.2 Approaches Based on Standards for Power Systems

4.2.2.1 IEC Standards and Ontologies

Approaches based on **IEC standards and ontologies** are described by Yang *et al.* (2020), as well as in other related works previously published by the same researchers (YANG *et al.*, 2017; YANG; VYATKIN, 2017; VOINOV *et al.*, 2017; ZHABELOVA *et al.*, 2014). These works address the transition from informal representation of requirements to formalized specifications. The approach is contextualized within standards like the IEC 61850 and IEC 61499, incorporating requirements modeling into the engineering process and using requirements to model and generate IEC 61499 control systems from IEC 61850 specifications. The control specification model is created in a form of ontologies, and the generation is based on ontology transformation. These works are related to D-SPADES in the sense that they largely apply MDE concepts, like DSLs and model transformations, however, their focus is on the substation automation domain.

4.2.2.2 CIM-based

Approaches based on the **IEC 61970** Common Information Model (CIM) and **IEC 62541 /** OPC Unified Architecture (OPC-UA), like the *CIMbaT*, described by Rohjans *et al.* (2011), which is a plug-in developed for the Enterprise Architect (EA) modeling and design tool.

CIMbaT exports the IEC 62541 / OPC-UA address space definition, in XML, from an IEC 61970 CIM compliant model, which is described in UML. CIMbaT is not an application development approach, but rather an automated mapping of a CIM model into the OPC-UA address space. The work is considered relevant in this context since, besides using a model-driven approach, it deals with standardized models and protocols (CIM and OPC-UA).

Another study mentioning similar model-to-model transformations is (GÓMEZ *et al.*, 2018), in which a power network model described in the IEC 61970 CIM is used for deriving Modelica models of physical power systems for dynamic simulations.

Interoperability standards like IEC 62541, 60970, and 61850 are also regarded as key technology by Lopez *et al.* (2010). In that work, a unified platform supporting the merging of network models and device models is proposed, such that enterprise-wide applications can be developed for the utility by means of these interoperability standards.

4.2.3 Smart Grids and Other Power Applications

4.2.3.1 Rapid Prototyping of Smart Grid Applications

Approaches focused on **rapid prototyping of smart grid applications** using DSLs are described in a series of papers apparently originated in the same research group, since they have some authors in common, including references: (ZANABRIA *et al.*, 2019; USLAR *et al.*, 2019; ZANABRIA *et al.*, 2017; ANDRÉN *et al.*, 2017; ANDRÉN *et al.*, 2013; ANDRÉN *et al.*, 2014; ANDRÉN *et al.*, 2014; ANDRÉN *et al.*, 2013; ANDRÉN *et al.*, 2015; ZANABRIA *et al.*, 2016; ANDRÉN *et al.*, 2016; DÄNEKAS *et al.*, 2014). Significant highlights from these researches include the definition of a DSL named Power System Automation Language (PSAL) (ANDRÉN *et al.*, 2017) and an ontology named Energy Management System Ontology (EMSOnto) (ZANABRIA *et al.*, 2017), which allegedly can be combined (ZANABRIA *et al.*, 2019). The PSAL language provides code generation support, assisting in the design of applications for layers of the Smart Grid Architecture Model (SGAM)[4], using a textual format. On the other hand, EMSOnto can generate informational reports on, for instance, controller conflicts. EMSOnto is intended to support control engineers during the conception, prototype, and implementation of control applications with a focus on multi-functional storage systems. A

[4] The SGAM is a structured approach for modeling smart grid use cases. For details, see (BRUINENBERG *et al.*, 2012)

main outcome of the ontology-based approach is the automatic detection of inconsistencies at the conception level. These works, similarly to the ones in the previous category, are related to D-SPADES with regard to the utilization of MDE concepts, although their application domain is focused on smart grid applications.

4.2.3.2 FMDE4SGRID

FMDE4SGRID, proposed by Felix *et al.* (2020), is a framework based on Model Driven Engineering and the weaving of models to support the development of inter-operable applications for the smart grid domain. The approach provides a DSL based on concepts from the smart grid domain, like the SGAM, and application integration based on the IEC 61970 (CIM) standard. Code generation is performed using the Acceleo tool at three different stages: (i) generation of Interface Definition Language (IDL)[5] files, (ii) generation of the configuration files, (iii) and the generation of Java source code.

This work, although focused on smart grid applications, has some remarkable similarities in methodology with D-SPADES, including:

- Explicitly addresses interoperability between devices and systems through standardized data formats and protocols, although it is not focused in SCADA/EMS applications.
- Adopts a modern MDE approach, describing the metamodel for the proposed DSL, based on the SGAM.
- Transformations are explicitly defined based on these metamodels, including Model-to-Model (M2M) and M2T.
- EMF and the Acceleo tool are also used to partially generate 3GL source code for an application.

The DSL definition, on the other hand, is based on Object-Oriented (OO) concepts and UML notation, which are not domain-specific languages for the power systems area.

[5] IDL is an OMG standard consisting of a descriptive language used to define data types and interfaces in a way that is independent of the programming language - https://www.omg.org/spec/IDL/4.2/About-IDL/

4.2.3.3 ThingML+

Moin (2021) proposes **ThingML+**, which extends the ThingML (HARRAND *et al.*, 2016)[6] with Data Analytics (DA) and Machine Learning (ML) techniques to facilitate the development of smart IoT services and CPS applications. It includes a methodology, a modeling language with extended syntax and semantics for DA and ML, as well as a number of code generators, based on ThingML. The idea with ThingML+ is that software models become capable of supporting ML, e.g., by generating the appropriate ML models for the task at hand and training them automatically using existing and/or incoming data. The approach, like D-SPADES, is also built on top of the EMF. Validation and evaluation are performed on cases from the smart energy systems and energy stock market domains.

4.2.3.4 Power-Attack

The **Power-Attack** modeling and simulating environment (CHHOKRA *et al.*, 2021) is focused on evaluating the effect of cyber attacks on the power system. One of its components is a domain-specific language that can be used to create scenarios involving faults in power equipment as well as cyber attacks on protection systems, controllers, and sensing data. The Power-Attack DSL is an imperative language that defines the simulation parameters and attack scenarios in the form of an attack model file. Such scenarios are evaluated in run-time by a simulation engine, in a sequence of steps to simulate attack scenarios. This approach, although being related to D-SPADES in regard to the use of DSLs in power systems, is not intended to produce executable SCADA/EMS software.

4.2.4 Approaches Based on Control Theory

4.2.4.1 MPC-based Algorithm

An **Model Predictive Control (MPC) based algorithm** is proposed by Beus and Pandžić (2022) for the design of the hydropower plant's primary frequency controller. With such an algorithm, the controller's internal linear prediction model parameters are continuously

[6] ThingML includes a modeling language and tool designed for supporting code generation and a highly customizable multi-platform code generation framework for embedded systems and Internet of Things (IoT) applications.

updated depending on the unit's operating point. The algorithm was implemented in a PLC and validated on a laboratory hydropower unit. No details are provided about the software development process though.

4.2.4.2 Event-Triggered DG Control

Event-triggered Distributed Generators (DGs) control proposed by Wang *et al.* (2019) which is a distributed, event-triggered, and time-decoupled secondary control for frequency/voltage restoration and power sharing in islanded microgrids. It is implemented as a Multi Agent System (MAS) consisting of a cluster of computers equipped with a Remote Procedure Call (RPC) framework. Each cluster node hosts an agent which is programmed to communicate with other agents and calculate the proposed event-triggered algorithm. The control algorithm consists of an event-driven control law, expressed as a discrete summation. No further information is provided about the software development process, though.

4.2.4.3 The DFR Algorithm

The **Distributed Frequency Regulation (DFR) Algorithm** described by Nazari *et al.* (2020a), Nazari *et al.* (2020b) addresses the LFC problem through a distributed approach. The algorithm is allegedly scalable either to small-scale prosumers, such as micro-grids, as well as large-scale control areas, such as the AGC system of utilities, thus it can be considered a SCADA/EMS application. The objective of the DFR algorithm is to maintain frequency stability and fairly optimal power sharing among generators, which are functionalities of traditional centralized AGC applications. The aforementioned papers describe the DFR algorithm for each participating agent as a set of steps, for which the corresponding equations are provided. Results for a set of simulations are described, however, no details are provided about how the controllers are implemented and simulated.

4.2.4.4 Classical Control Theory

Approaches based on **classical Control Theory** are common in several studies describing models for load/frequency control or voltage control applied to bulk generation or transmission systems, which are typically SCADA/EMS applications, and therefore our main

area of interest. The model representation used in these works typically includes block diagrams, state space models, differential/integral equations, s or z domain equations, or a combination of those. These works concentrate on presenting the application's model and the results of tests or simulations.

Typically little detail is provided about how the models are transformed into software artifacts or about the nature of such artifacts. In other words, the software development process for the application is not described in detail, and the project's design/modeling phases are apparently decoupled from software development. On the other hand, the ubiquitous presence of the block diagram and transfer function representation in these works is a clear hint that an adequate DSL for this domain must fully support such abstractions. Publications reviewed include: (LOU *et al.*, 2019; AZIZI; KHAJEHODDIN, 2018; ARYA; KUMAR, 2017; HARVEY *et al.*, 2017; BAEK, 2014; BAEK *et al.*, 2013; BEVRANI *et al.*, 2012; MARTÍNEZ *et al.*, 2012; BEVRANI; HIYAMA, 2009; SHENG *et al.*, 2009; SUN *et al.*, 2009; ROBERT; HURTADO, 2008; SHAYEGHI; JALIL, 2007; CORSI *et al.*, 2004a; CORSI *et al.*, 2004b).

4.2.4.5 NewSART Project

The **NewSART project**, described by Sulligoi *et al.* (2011) is a particular work in which the modeling is largely based on the Control Theory, and a fairly detailed description of the modeling and transformation process is provided. NewSART performs Secondary Voltage Regulation (SVR) as part of the hierarchical control system of a high-voltage transmission network. Control and communication functions are implemented using open source tools, including *Scilab/Scicos* environment to create a mathematical model that is automatically transformed into textual artifacts (C source code) for the *Linux RTAI* platform.

The work shows some features similar to D-SPADES:

- It originates in the same application domain, which includes voltage and reactive power control, although the integration with the SCADA platform is not clearly addressed. D-SPADES, on the other hand, explicitly addresses the integration with SCADA.
- AO models are used as source models, although it is not clear which parts of the system are modeled using this approach, and which parts are manually developed. D-SPADES, on the other hand, is capable of producing fully functional applications without the need for manual coding, although partial generation of modules for legacy applications is also

supported.

- The transformation process, although automatic, is performed by means of code generators available in the *Scilab/Scicos* environment. Essential concepts of MDE approaches, like DSLs, metamodeling, and transformations are not mentioned in the text: Sulligoi *et al.* (2011) do not explicitly regard its approach as MDE. In other words, code generation is performed through a "black box" approach. D-SPADES, on the other hand, proposes a DSLs and adopts a template-based approach to code generation, which presents several advantages further discussed in Section 5.6.

- The target software artifact of the process is 3GL source code, in the C programming language. It is not clear whether other languages can be used. Comparatively, D-SPADES's template-based approach is targeted to C++ programming language, but can be easily adapted to other languages by using different templates. In fact, we have already developed a template for generating FSMs in Fortran, to be integrated into legacy applications developed in that language. Such templates and applications are further discussed in Chapters 5 and 6.

4.3 ANSWERS TO THE REVIEW'S RESEARCH QUESTIONS

Based on the reviewed studies, we can outline the answers to the research questions in the scope of the literature review, as discussed below:

RQ1: *What are the overall characteristics of the software process being applied to the development of power systems applications?*

Apparently few publications explicitly deal with this problem, however, we were still able to identify the following characteristics:

- The most common **source models** being used for design and specification include:

  1. block diagrams associated with mathematical models from control theory;
  2. modeling languages related to the AO paradigm, i.e.: based on a block-diagram notation, like Matlab/Simulink/Stateflow, Modelica, Scilab/Scicos and the *Ptolemy II* (ZHABELOVA *et al.*, 2014);
  3. models standardized by the IEC (61850, 61499/61131, 61970, and 62541);

4. models already consolidated in the areas of systems and software modeling like UML, Systems Modeling Language (SysML)[7], MARTE or other UML profiles;

- Regarding the **software artifacts** produced from the source models, although several of the reviewed studies do not specify this information, the most frequently mentioned types include:

  1. the IEC 61499/61131 executable format;
  2. textual artifacts, including source code;
  3. platform-specific, legacy, or other proprietary formats;

- Regarding the **transformation process** of the source model into other software artifacts, many of the reviewed studies do not describe how this task is performed, although a few mention a manual transformation process. A significant number of studies, however, mention the use of automatic code generation, including through M2T transformations. The supporting environment typically used for this task is the Eclipse EMF.

RQ2: *Is the concept of MDE being applied to the development of SCADA/EMS, or other power system's applications?*

Judging by the search results obtained from the DLs, we can infer that, so far, few studies have investigated the MDE approach to the development of SCADA/EMS applications. Although we have observed a significant number of studies mentioning MDE-related concepts, only a few of them are specifically concerned with EMS. Relevant information obtained from the review includes:

- regarding **domain-specific languages**:

  1. SCADA/EMS applications are frequently modeled using control theory concepts, and although such models are widely accepted among power systems specialists, they cannot be considered software modeling languages.
  2. The modeling approach described in the IEC 61499/61131 can be used to specify a fully functional application, however, it does not incorporate domain-specific concepts related to the power systems domain. Another limitation of this approach is that it is typically targeted at PLC platforms.

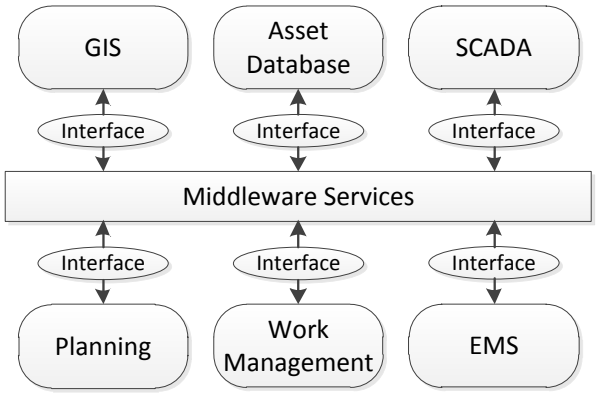
[7] https://sysml.org/

3. Similar observation can be made about the UML-based languages: they can specify software structure and behavior, but are not specific to the power systems domain.
4. On the other hand, formats such as those proposed by IEC 61850 and 61970/61968 standards are specific to the power industry, however, restricted to certain aspects of the system model, like communication and information model, thus lacking the expressiveness needed for specifying functionality and behavior.
5. Languages associated with modeling and simulation environments, such as Matlab/Simulink/Stateflow, Modelica, Scilab/Scicos, and the *Ptolemy II*, integrate both a visual schematic and equation-based description of the system under study, therefore are suitable to accommodate concepts specific to SCADA/EMS applications, like the control theory.

- in terms of **transformation engines and generators**, we have observed references to the open-source Eclipse Platform tools, including the Eclipse Modeling Framework and the Atlas Transformation Language. W3C XML-based technologies, the OMG Query Views Transformations, and the Semantic Web Rule Language were also mentioned (YANG *et al.*, 2017). Other specific techniques such as ontology transformations and custom model-to-model transformations were observed. Proprietary or tool-specific code generators, like those supported by design environments such as Matlab and Scilab, are also mentioned.

- for **integration** with commercial SCADA packages, many publications mention the IEC 61970/61968 EMS API, which defines the Enterprise Service Bus architecture of Figure 24, although such architecture constitutes only a conceptual model, i.e.: the standard does not prescribe specific technologies for its implementation. The IEC 61850 station-level interoperability definition, based on the ISO 9506 Manufacturing Message Specification (MMS) protocol is also mentioned. The viability of such technologies needs further evaluation since none of the reviewed studies describe actual SCADA/EMS applications integrated through them. The IEC 62541 - OPC-UA standard, on the other hand, is mentioned by some studies along with instances of real applications.

Figure 24 – Enterprise Service Bus model for inter-application communication.



Source: Neis *et al.* (2019), Electric Power Research Institute (2018).

4.3.1 Open Problems

In the set of studies reviewed in this work, the generation of complete applications using the MDE approach is rarely seen. Most of the studies mentioning MDE-related concepts describe the generation of part of the system, like an IEC 61850 specification or a simulation model. Particularly in the case of SCADA/EMS applications, we were unable to identify concrete instances where the MDE approach has been explicitly applied to the development of full applications, except possibly by the work of Sulligoi *et al.* (2011), which asserts that "from the simulated mathematical model, a C source code is automatically generated". On the other hand, the MDE approach is apparently being adopted for automation and distributed smart grid applications. Therefore, the viability and advantages of this approach to SCADA/EMS applications development are yet to be determined. Some of the open problems identified in the previously mentioned studies, which D-SPADES proposes to address include:

- Although some studies clearly mention the adoption of MDE approaches to certain power system's applications development, a systematic MDE approach to EMS has not been found in the literature, as the data presented in Table 2 suggests.

- The types of models (languages) employed by power systems specialists to represent EMS applications include concepts specific to the domain, like the block diagram and transfer function notations. Generic software modeling languages, like UML, are not compatible with such paradigms.

- Although specific methodologies for modeling data/information in the power sector exist, for instance, the CIM and IEC 61850, such models are not appropriate to represent

system/software behavior.

- As a consequence, the EMS development process is apparently performed in two different stages[8]: (i) first the power system specialists model and validate the desired behavior using their domain-specific tools and notations; (ii) later this specification is passed along to the software development team, which manually translates the models into programming languages and other types of models. Such a process becomes error-prone and hinders future system evolution since requirements changes necessarily involve manually changing the software. Additionally, the evolution of such systems is often performed directly in the application's source code, with a tendency to leave the high-level models outdated (SCHMIDT, 2006).

- It is not clear what is the role of the power system specialist in the development process, and how his/her modeling activities are interfaced with the software development team. The benefits of having the artifacts designed and tested by the power system specialists and later automatically transformed into software implementation are not being fully explored.

- Some of the reviewed studies address the problems of interoperability and integration of applications using standardized and consolidated technologies, however not for EMS applications. We have not been able to find studies that explicitly address how to develop new applications and integrate them with COTS SCADA packages.

4.4 REVIEW CONCLUSIONS

We believe that MDE approach may bring significant improvements to the SCADA/EMS applications development process, based on the following observations:

- makes it possible to reuse artifacts produced during early specification and design phases through the whole development process, by performing the transfer and automatic model transformations between modeling phases (HÄSTBACKA *et al.*, 2011). Ideally, the same artifacts used by power systems engineers to specify, design and validate the control system behavior should be transformed into software components;

[8] We reinforce this statement based on our experience working in the power system's area for more than 20 years. Such a conclusion is corroborated by the data presented in Table 2, although the literature review itself cannot demonstrate the statement with hard evidence.

- it can allow the power system's control specialists to actively participate in the development process, using domain-specific, but platform-independent modeling languages;
- by being platform independent, the DSL may facilitate the porting and migration of applications from one platform to others;
- utilities, SCADA/EMS vendors, service providers, and integrators can all benefit from those concepts, applying MDE methodology to more easily design, develop and deliver customized solutions.
- ultimately MDE approach may cut development costs and time, reduce defects, improve software quality, maintainability, and requirements traceability.

Thus, the MDE concept has the potential to make the whole SCADA/EMS application development process more agile and less error-prone. We were also able to highlight some of the desired features and promising technologies for SCADA/EMS application development identified in the literature review:

- The use of block diagrams and transfer function notations to express models is almost ubiquitous. Therefore, we conclude that a good DSL for this domain must natively support these abstractions.
- Some of the reviewed works are targeted at automation platforms based on PLCs, thus the types of artifacts produced include PLC programs in the IEC 61499/61131 format. This format is not appropriate for SCADA/EMS applications, since they are typically not targeted at PLC platforms. The appropriate format, in this case, includes 3GLs like C/C++, which are also mentioned in some of the reviewed studies.
- A good, open-source tool supporting MDE methodologies is the Eclipse EMF, which is mentioned by some of the reviewed works.
- The International Electrotechnical Commission (IEC) standards for integration, like IEC 62541 / OPC-UA is mentioned by some of the reviewed works. OPC-UA is a platform-independent standard through which various kinds of systems and devices can communicate through messages between clients and servers or between publishers and subscribers over various types of networks.

5 THE D-SPADES APPROACH TO EMS SOFTWARE DEVELOPMENT

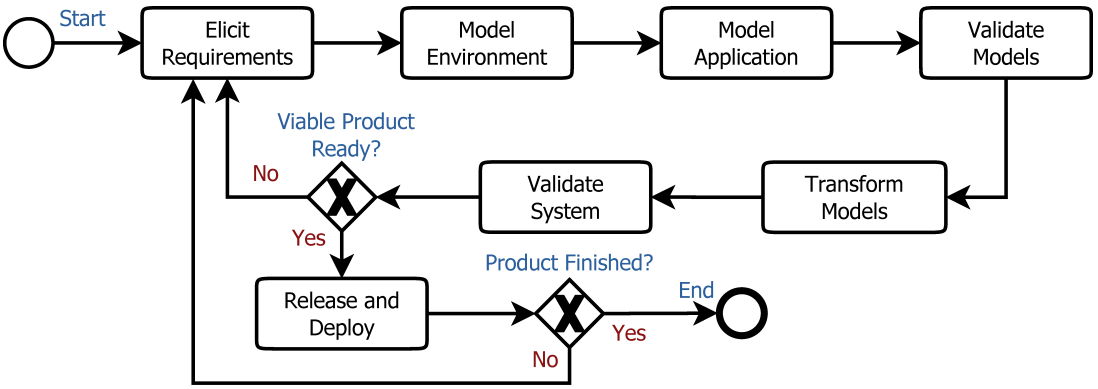
In this chapter, we describe D-SPADES: our proposed MDE-based approach to the development of EMS applications. According to the definition given in Section 3.3.3, and in the context of hydropower plants operation, we define "EMS applications" as the suite of centralized or off-site functions running on top of a SCADA layer. Such functions include, but are not limited to: automatic generation control, voltage and reactive power control, production/interchange scheduling, operator assistance, and analysis tools. Although the development and validation of D-SPADES was focused on this specific category of applications, virtually any application running on top of the SCADA layer could be developed through D-SPADES.

D-SPADES is established upon the following components: (i) an actor-oriented **language** and platform for modeling hydropower plant applications; (ii) the mapping strategies for **model transformation** from actor-oriented to object-oriented (or even procedural) programming models; (iii) the necessary **tool support** for the approach, including modeling tools, transformation languages, components library, and SCADA integration API; and (iv) the **software process** that coordinates the software production through D-SPADES approach. A full paper describing D-SPADES is accepted for publication in the International Journal of Electrical Power and Energy Systems (NEIS *et al.*, 2023).

5.1 D-SPADES SOFTWARE PROCESS

The D-SPADES software process is essentially composed of the activities listed below. These activities are executed in an iterative and incremental fashion throughout the whole software life cycle, as schematically represented in Figure 25. Each one of the process activities is briefly described in the following sections. As further discussed in Chapter 6, we have applied this process to the development of real-world applications at the Itaipu power plant. Therefore, the examples from Chapter 6 further illustrate and clarify the application of the D-SPADES process, and are complementary to the description provided in the current chapter.

Figure 25 – D-SPADES Software Process Model.



Source: The author.

5.1.1 Problem Characterization and Requirements Elicitation

This phase involves the problem analysis and the collection of requirements for the application under development. Requirements can be collected from several sources, for instance: through observation of existing systems, discussions with stakeholders, document analysis, etc. Eliciting the requirements may involve the development of one or more system models and prototypes, which can be expressed using the EMSML language. In the power plant scenario, two groups of applications can be identified, for which different sources of requirements may exist:

1. **Legacy applications** that are being improved, migrated or re-implemented using D-SPADES approach: Existent applications being evolved or migrated to a different SCADA platform using the D-SPADES approach have a set of artifacts already available that can leverage the development process. For instance, existing documentation and models, like block diagrams and transfer functions, can be re-used. Legacy source code can also be used for detailing and refining the models, as well as for populating the D-SPADES components library.

2. **New applications** developed using D-SPADES approach: Specification of new functionalities can start, as early as possible in the project development, by describing actor-oriented models, even when these models are early sketches. Such models can be incrementally validated, evolved, and later transformed into the target software artifacts. The model construction activities can also be made directly into the actor-oriented modeling environment adopted by the approach, in our case the *Ptolemy II* tool. Interaction with project

stakeholders may be facilitated by the use of the domain-specific EMSML modeling language.

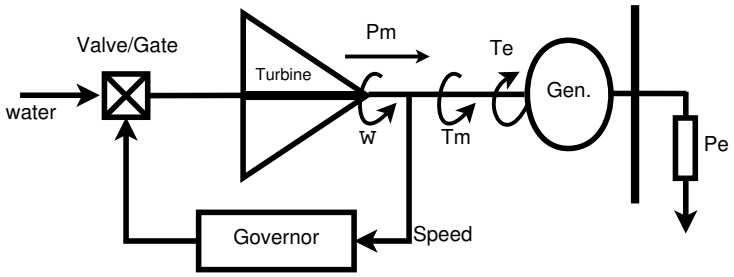
As an illustrative example, consider a hypothetical scenario where a new generating unit's speed-governing function needs to be developed. We acknowledge that speed governing is not a typical EMS application: it is in fact part of the Unit Local Control, according to the IEE Std. 1249 (IEC/IEEE, 2013), as discussed in Section 3.3. We believe, however, that this example provides an instructive demonstration of the D-SPADES process, since the concepts involved in the primary regulation have already been discussed in Section 3.2.1.2 (page 50). As requirements for this governor, consider that:

1. It must be a discrete governor. The necessary signals from the physical system are already provided as uniformly sampled with a 4 seconds periodicity. Control actions will also be performed at this periodicity.

2. It must be an isochronous governor, i.e., it must have no steady state speed error.

Figure 26 schematically represents the physical process and the speed governor component.

Figure 26 – Discrete speed governor example.



Source: The author

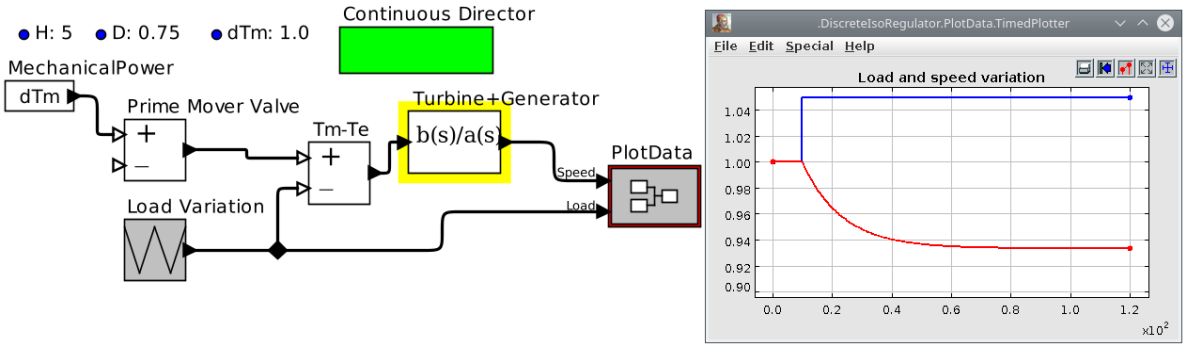
5.1.2 Environment Modeling

Consists in building a model of the cyber-physical environment, including processes and other applications to interact with. This environment model is used to validate the application's requirements and its model's design. These activities can be leveraged by existing models of the CPS which were previously developed, such as: generating unit's governor and excitation system models, electrical network models, protective devices models, and so on.

For our speed governor example, we must build a model of the physical process and its interfaces with the application. In this case, let's assume that the system representing the

turbine/generator/load has a typical transfer function of the form $1/(2 \cdot H \cdot s + D)$, with $H = 5$ representing the inertia and $D = 0.75$ representing the load damping factor. Figure 27 represents a screenshot of this example environment being modeled on the *Ptolemy II* suite, and simulated in open-loop (no governor). In this case, the prime mover mechanical power is modeled as a constant value (1 pu), while its control valve is modeled as an add/subtract block, meaning that its resulting power can be modulated by an input signal subtracted from the constant value. A load variation is modeled as a step function, increasing from 1 pu to 1.05 pu after 10 s of simulation, as shown in the blue trace. As a result, the output speed is reduced and stabilizes at an off-nominal value due to the effect of the load damping.

Figure 27 – Environment model for governor example.



Source: The author

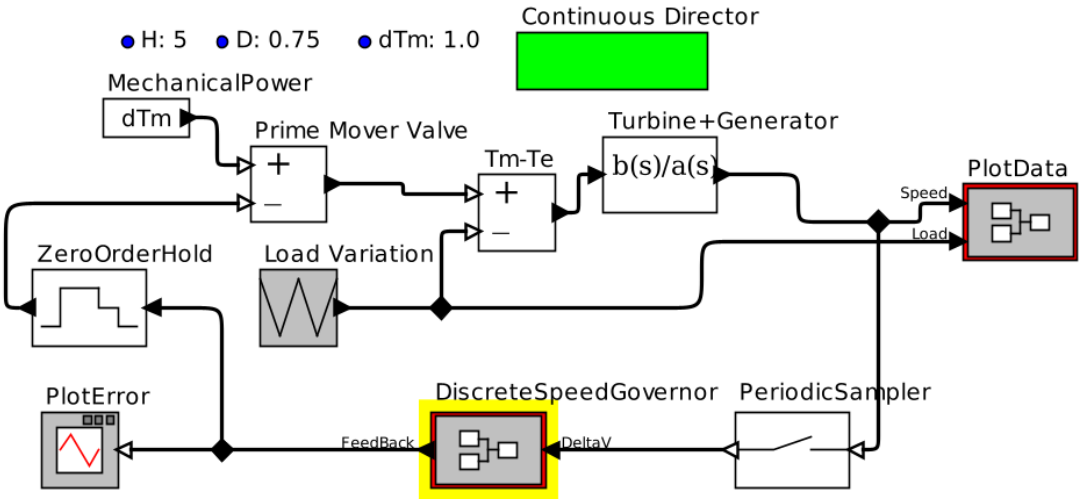
5.1.3 Application Model Design and Construction

The design and construction activities are focused on evolving the model specification from previous phases into fully functional models. Models and specifications previously used for studying and planning other aspects of CPS may also serve as input. These activities are heavily dependent on the actor-oriented modeling environment. They may also involve domain specialists in the process: people with skills in power system control and operations, but not necessarily trained in software development.

Returning to our illustrative example, we would be modeling the actual discrete speed governor. To the above system, our environment model must also include the periodic sampler and the sample-holder components that interface the continuous part of the CPS with the discrete governor, as illustrated in Figure 28.

Finally, the discrete governor, highlighted in yellow in Figure 28 must be designed. In this case, we designed it as a simple proportional-integral controller, composed of a gain and an

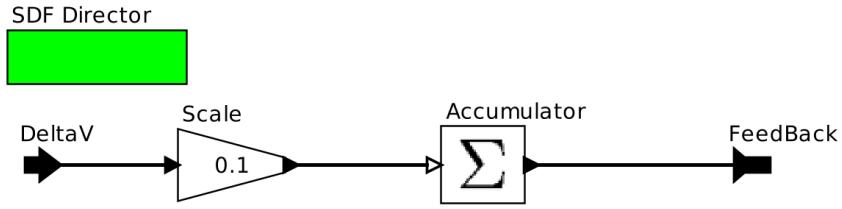
Figure 28 – Environment and Application model.



Source: The author

accumulator, as shown in Figure 29, thus ensuring that it behaves as an isochronous governor. This model is executed under the SDF MoC, thus operating like a typical signal processing application triggered by incoming periodical samples.

Figure 29 – Simplified discrete governor model.



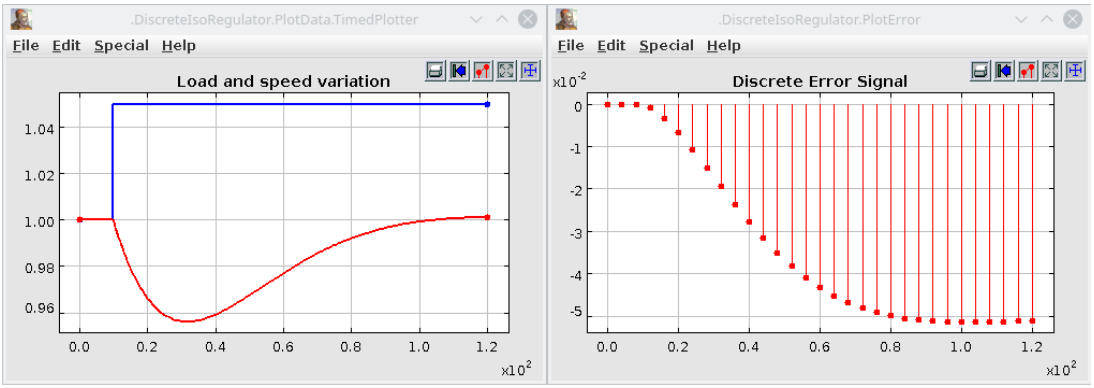
Source: The author

5.1.4 Model Testing and Validation

Model testing and validation rely heavily on the simulation capabilities of the *Ptolemy II* actor-oriented modeling environment. *Ptolemy II* can model and simulate both the EMS application under development, as well as the corresponding physical process associated with it. The simulations shown in Figures 27 and 30 illustrate this capability: the first case illustrates the open-loop simulation of the system's behavior, while the last demonstrate the resulting behavior in the presence of the proposed controller. We can see that the speed was brought back to 1 pu by means of the discrete control actions performed by the governor, therefore validating the proposed design. Additional validation could be achieved through co-simulation of other CPS components, performed by integrating *Ptolemy II* with other simulation environments, such as

the Itaipu's DTS (see Section 3.4.4, page 71). Details of how to perform such co-simulation, though, are out of the scope of this research.

Figure 30 – Simulation results for governor operation.



Source: The author

5.1.5 Model Transformation

This activity involves applying the MDE tools to automatically generate software artifacts, including source code, from the models produced through the previous activities. It also involves inspecting the artifacts and building the executables for the target SCADA platform. At the current stage of development, D-SPADES is able to produce source code directly from AO models using the supported MoCs. Thus, every aspect of an EMS application that can be expressed as an AO model (e.g. signal processing, closed-loop control logics, and determination of the necessary output actions) can have its code automatically generated. Other aspects, such as user interfaces, are not currently supported by the approach and must be performed using the vendor-specific tools provided by the SCADA environment being used. Therefore additional development activities might be necessary during this phase, in which such artifacts are constructed using conventional approaches.

In our speed governor example above, the model transformation activities consist in automatically producing source code from the *DiscreteSpeedGovernor* block highlighted in Figure 28, and building the executables for the target platform This activity will be further detailed in section 5.3.

5.1.6 System Validation

System validation or “system testing” consists in integrating the executable software and other accompanying artifacts produced by the execution of earlier activities with the SCADA/EMS platform, and ensuring that the whole system works as expected. It is advisable to start this validation on a testing platform, with simulation capabilities, like Itaipu’s DTS. Although the simulation platform is known to have limitations, it helps identify problems early and gain confidence and experience with the application. Finally, the application must undergo controlled tests under actual operational conditions before it can be declared production-ready. Once the system test is successfully executed, the running software is ready to be deployed in the production environment.

Returning to our example above, this activity would involve performing tests in the execution platform of our application, which are typically PLC-based in the case of discrete speed governors. The physical system, i.e., the generating unit, could be simulated for preliminary tests, but controlled tests on the real system would be required.

5.1.7 Release and Deploy

Deploying an application means putting it in service, to be used in real operational processes (SOMMERVILLE, 2011). It involves the transfer of the artifacts previously validated into the production environment, which consists of the SCADA/EMS servers and workstations. In a power system, such activities are subject to rigorous norms and operational procedures established both at company-level as well as power-grid-level. It involves, for instance, obtaining work authorization from the Operations Department, as well as coordinating activities with external agencies like System Operators. Those activities are not described here, since they involve details specific to the environment where the application is deployed, including interactions with other business processes. In our hypothetical speed governor, the generating unit would probably need to be shut down in order to deploy the governor, therefore such operations would need approval from system authorities.

5.1.8 Further Considerations: Maintenance and Evolution

Software modifications may be triggered by changing business requirements, by reports of software defects, or by changes to other parts of the CPS environment (SOMMERVILLE, 2011). For instance: an upgrade to the SCADA platform may result in changes to the EMS application layer; or the commissioning of new electro-mechanical equipment may require changes in the control loops. Once such a need for modification is identified, another instance of the D-SPADES process is started, using as inputs the artifacts previously developed for the application, as well as the new requirements. It is worth mentioning that the existing modeling artifacts can usually be re-used in future evolutions of the system, i.e., the modeling activities are not required to start from scratch.

In the case of our hypothetical speed governor, let's assume that a new requirement is established: it must now have a configurable proportional gain, externally accessible, for instance, by the operator. Figure 31 illustrates how this behavior could be modeled, i.e., by providing the desired gain as an input to the governor block. As we can see, modifying the gain from a fixed "0.1" value to "0.3" changes the system's response: it moves faster towards the nominal speed but shows some overshoot as a collateral effect.

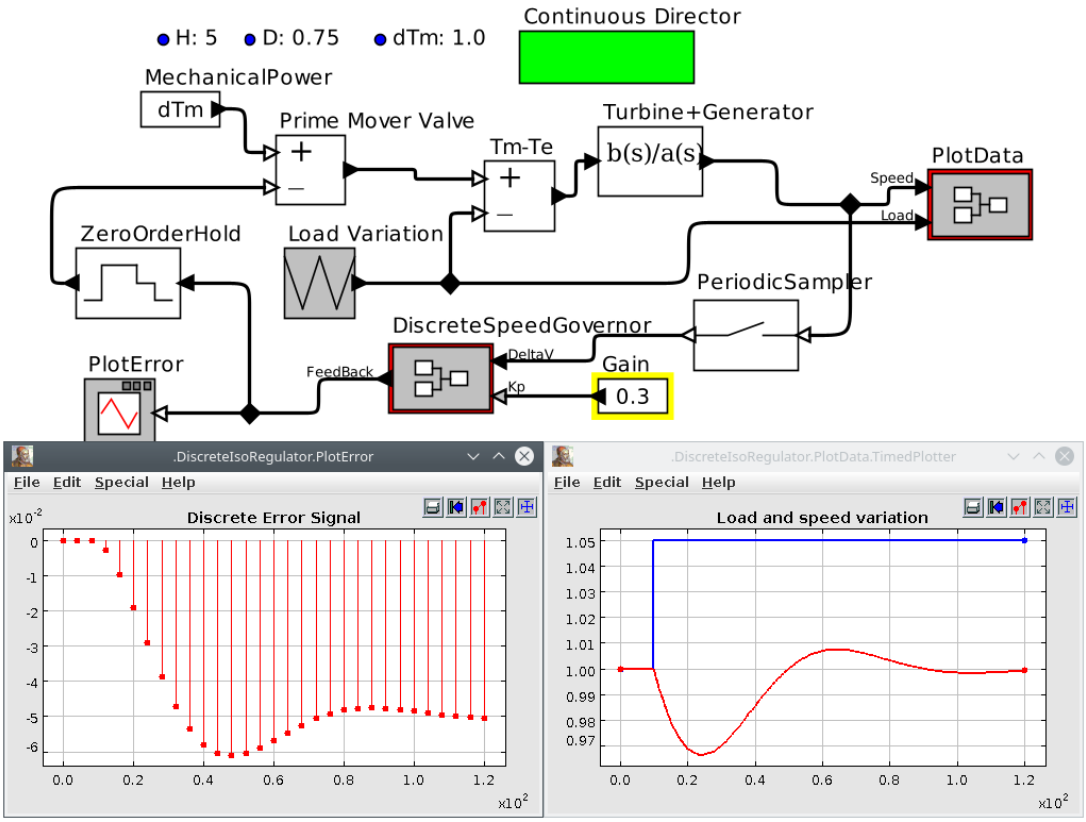
5.1.9 Simplified Workflow

Figure 32 shows a simplified view of the activities and artifacts involved in developing an executable EMS application through D-SPADES. This basic workflow includes the following activities:

1. *Model Building* addresses the development and testing of models;
2. *Model Conversion* consists in converting the file format used by the modeling tool into a standard file format known as XML Metadata Interchange (XMI)[1];
3. *Model-to-Text (M2T) transformation* processes the model and automatically generates the source code implementing the corresponding behavior in a 3GL;
4. *Compiling and Building* the executable application using the D-SPADES components library and the SCADA integration libraries.

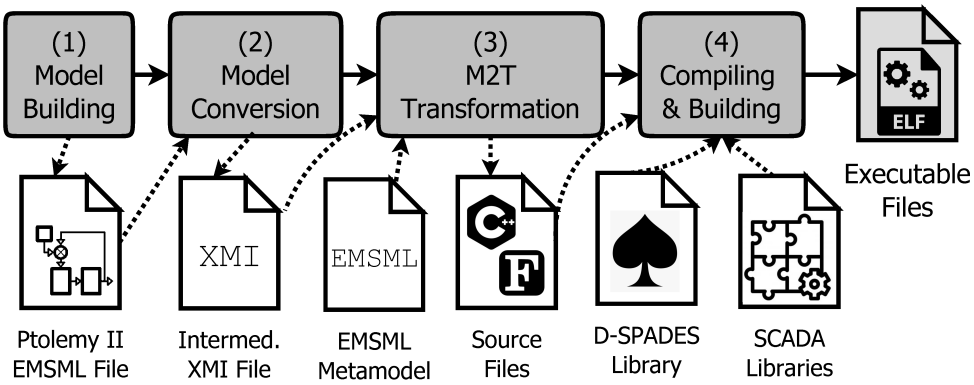
[1] https://www.omg.org/spec/XMI/

Figure 31 – Adding new governor requirement.



Source: The author

Figure 32 – D-SPADES workflow and produced artifacts.



Source: Neis *et al.* (2023).

5.2 A MODELING LANGUAGE FOR EMS APPLICATIONS

Based on the information gathered from our literature review, we have concluded that a language supporting the block diagram and transfer function paradigms, among other useful constructions, would be appropriate to model EMS applications. In this review we were unable to identify any existing integrated development environment supporting such paradigms, readily applicable to the modeling of EMS application, and possessing the desired properties (e.g. that

could be automatically processed and transformed into source code for an arbitrary platform). Therefore we propose to adopt an AO language for D-SPADES, which is described in the following sections. The D-SPADES approach however is not necessarily limited to this single language: other types of models, including legacy designs and simulations built for different modeling environments, can be supported. This support can be implemented either by creating M2M transformations from the source/legacy models to EMSML, or by rewriting the M2T templates that process EMSML in order to process a different modeling language. In either case, the metamodel of this alternative language needs to be known.

According to the guidelines from (BRAMBILLA *et al.*, 2017), language design through metamodeling is a three-step iterative and incremental process composed of:

1. **Modeling domain analysis:** defines the *purpose*, *realization* and *contents* of the language. This includes the analysis of the modeling domain, which in our case is the hydropower plant EMS applications described in Chapter 3. Some examples of applications that can be modeled using EMSML include JBVRC (see Sec. 3.4.3.2, page 70) and AGC (see Sec. 3.4.3.1, page 69). The requirements of EMSML language will be defined iteratively based on such applications.

2. **Modeling language design:** formalizes the modeling concepts by modeling EMSML's abstract syntax, i.e., defining its metamodel.

3. **Modeling language validation:** consists in using EMSML to model the reference examples, validating the completeness and correctness of the metamodel.

In practice, the complete process also includes the adoption or development of a concrete syntax, so it allows us to interact with the domain experts.

5.2.1 Domain Analysis

According to Brambilla *et al.* (2017), the domain analysis can be done, for instance: (a) by abstracting recurring patterns found in legacy source code into modeling concepts; (b) by document analysis and/or interviews with experts to derive modeling concepts. In the case of EMSML we will apply a mix of these strategies, but initially we are performing the analysis primarily based on existing documentation and published works.

As verified in our literature review (NEIS *et al.*, 2019), a very common paradigm for modeling EMS applications is the use of traditional control theory models, which include block diagrams and their corresponding transfer functions. Models built on software tools such as Matlab/Simulink, Modelica and *Ptolemy II* could be described as a software implementation of control theory paradigms, thus figuring among the languages being used for power systems modeling. These tools are categorized as "actor-oriented modeling tools" since they use "*block diagram based design environments and the design usually starts with assembling preexisting components in the library*" (ZHOU *et al.*, 2007). If we assume the definition of "actor" to be *an encapsulation of parameterized actions performed on input data that produces output data* (ZHOU *et al.*, 2007), then the blocks and transfer functions paradigms from control theory can be easily accommodated as actor-oriented models. Therefore we believe that the actor-oriented programming model is a suitable abstraction to represent EMS applications.

In the actor-oriented programming model, the concepts listed in Table 3 are identified, which can guide the design of EMSML metamodel. These concepts are a subset of the elements used in the MoML metamodel definition (LEE; NEUENDORFFER, 2000). A brief description

Table 3 – Modeling concepts for EMSML.

Concept	Intrinsic Properties	Extrinsic Properties
entity	name, class	Arbitrary number of *links, entities, relations, properties and ports*; At most one *director*
port	name, class	Arbitrary number of *properties*
link	port, relation	
relation	name, class	
director	name, class	
property	name, class, value, version	Arbitrary number of *properties*

and some examples of these concepts are presented below:

- **entity:** may represent a composite actor (containing other models), an atomic actor, or a state inside an FSM. Therefore, actors and states are subclasses of the entity class. In terms of the topology describing the model, an entity is a vertex in a generalized graph, and it also aggregates **ports**.

- **port:** is the interface of an entity to any number of relations. The role of a port is to aggregate a set of links to relations. Thus, for example, to represent a directed graph, entities can be created with two ports, one for incoming arcs and one for outgoing arcs.

- **link:** associates ports with relations, thus creating communication channels, i.e.: they keep track of which ports are connected to which relations.

- **relation:** relations are “splitters” that enable connecting two or more ports. The relation broadcasts the output from a single output port to any number of input ports. The relations thus represent connections between ports, and hence, connections between entities. Every single port has only one connection: a connection to the relation. In terms of object-oriented concepts, the relation implements a classical “mediator” pattern: it reduces coupling and simplifies communication between actors in a model. For instance, if the output port of a given actor is to be directed to two or more input ports in other actors, the logic responsible for routing the messages resides in the relation, thus simplifying the port implementation.

- **director:** A director governs the execution within a model, or a specific part of the model delimited by a composite actor. A composite actor that contains a director is said to be opaque, and the execution model within the composite actor is determined by the contained director. Composite actors that do not contain an instance of a director have their behavior governed by the director of its container. In terms of implementation, the director is responsible for invoking the actors contained by the model, or parts of the model. In the current version of D-SPADES, no actual implementation of the director class is necessary, since the set of MoCs currently supported is restricted to SDF and FSM. Future evolutions of D-SPADES might need to explicitly provide director implementations in order to support other MoCs.

- **property:** properties may define expressions and named parameters used in a model (e.g. a constant value like “$E_b : 500$” defining the system’s base voltage level, or a formula like “$E_{pu} = E/E_b$” defining the system’s per-unit voltage calculation); or certain characteristics of the container objects (e.g. the direction of data flow in a port).

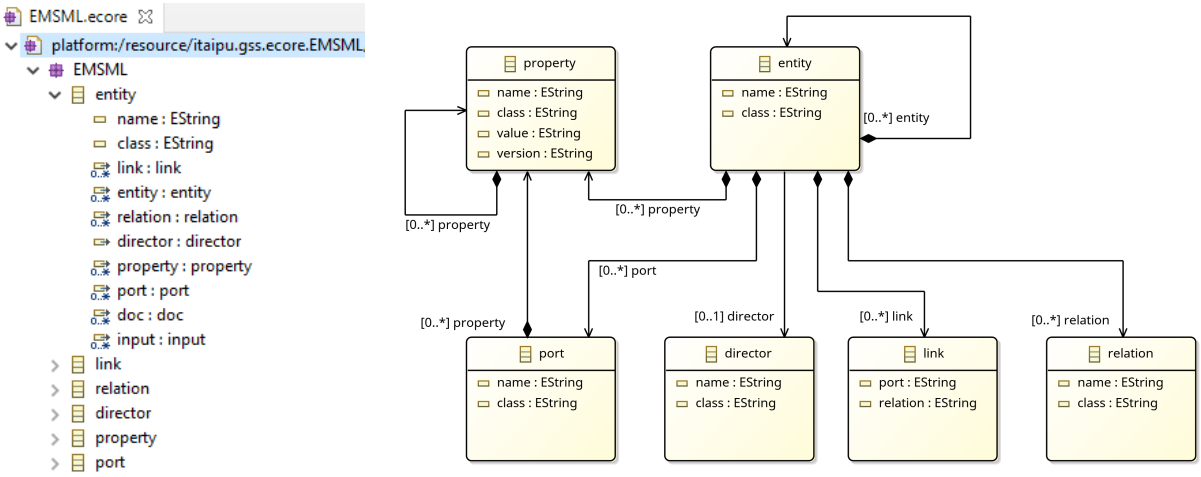
5.2.2 Modeling Language Design

The two most important ingredients of a modeling language are its *abstract syntax* and its *concrete syntax* (BRAMBILLA *et al.*, 2017). In model-centric languages, the *abstract syntax* consists of the definition of the metamodel, which contains the modeling concepts and their properties. The metamodel defines all valid models that can be expressed by EMSML.

We have built the EMSML metamodel (M2 level) shown in Figure 33 (a) using the Ecore language (M3 level). A corresponding UML-style representation of this model is shown in Figure 33 (b). The complete EMSML metamodel is shown in Appendix A. The elements

of this metamodel map to the concepts listed in Table 3 as follows: *concepts* are transformed into *EClasses*; *intrinsic properties* are transformed into *EAttributes* and *extrinsic properties* are mapped into *EReferences* between *EClasses*. This metamodel definition consists of the abstract syntax of EMSML. The classes and relationships shown in Figure 33 (b) can be traced back to the same concepts describing the MoML notation, shown in Figure 6 (see page 43).

Figure 33 – The Ecore metamodel and corresponding class diagram for EMSML.



(a) Ecore metamodel. **(b) Corresponding class diagram.**

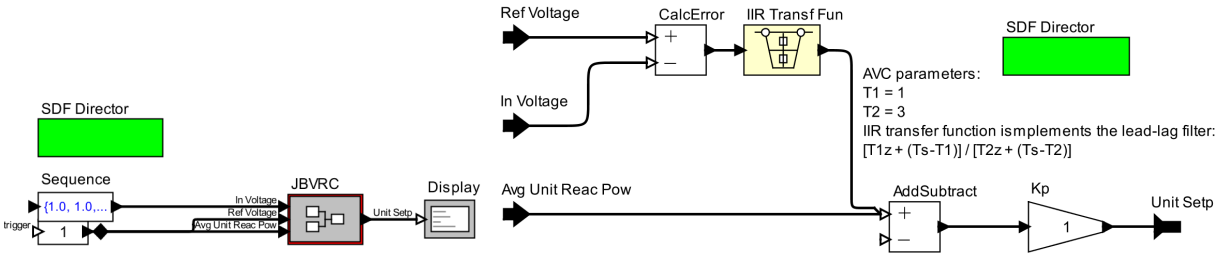
Source: The author.

The concrete EMSML syntax is implemented through an XMI[2] file, which corresponds to the serialization of the elements in a given model to an XML format. Listing B.1, shown in Appendix B, is an excerpt from an EMSML file, where some instances of the metamodel classes are used to describe a simple model. The XMI concrete syntax is an essential artifact for D-SPADES in order to perform model transformations. However, for the modeler, this textual syntax is not very useful. The modeler would rather use a graphical syntax to express models, such as the example shown in Figure 34. This image shows the same model from Listing B.1, rendered in the *Vergil* editor, which is part of the *Ptolemy II* package (BROOKS *et al.*, 2014b). Since the concepts modeled by EMSML consist in a subset of the the MoML (LEE; NEUENDORFFER, 2000) metamodel, the concrete syntax for EMSML can also be implemented in the *Vergil* graphical editor. The graphical elements in this picture correspond to the textual elements from Listing B.1, with a much richer visual representation.

Some of the most significant features distinguishing MoML and EMSML include the following:

[2] XML Metadata Interchange, an OMG standardized format for exchanging metadata information via Extensible Markup Language (XML) – https://www.omg.org/spec/XMI/.

Figure 34 – A toy application model illustrating EMSML's graphical concrete syntax.



(a) Outer model. **(b) Contents of composite JBVRC actor.**

Source: The author.

- MoML supports several graphical properties used for displaying purposes in the *Vergil* editor. Such features, although present in the EMSML metamodel, have no meaningful semantics and are ignored during the model transformations performed in D-SPADES.

- The D-SPADES components library currently is rather limited when compared to the *Ptolemy II* library. Some of the native *Ptolemy II* actors were implemented in D-SPADES, and all the new actors proposed for the D-SPADES library were implemented into the *Ptolemy II* library, although these D-SPADES-specific actors were not incorporated into the official *Ptolemy II* distribution. This difference refers to functionalities implemented by each library, rather than syntactic differences in the language.

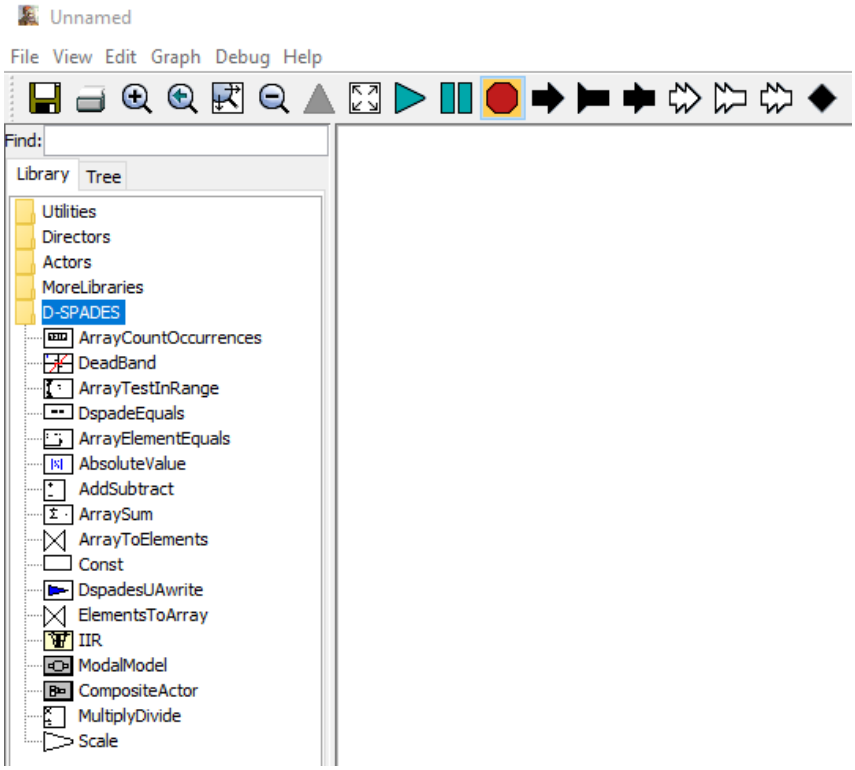
- The MoCs supported by D-SPADES are currently limited to the SDF and FSM, while *Ptolemy II* supports several other MoCs.

This distinction can be better characterized in the *Vergil* editor, for instance, by building a user library containing all the actors supported by D-SPADES, as shown in Figure 35. The set of actors available to the user for building models is clearly identifiable under the D-SPADES folder.

Figure 36 shows the graphical representation of some of the actors and other language elements used within both *Ptolemy II* and EMSML. A brief description of these elements is provided below. Notice however that a far greater number of actors exist in the *Ptolemy II* environment, of which only a certain subset is currently implemented in the D-SPADES library. New actors can be easily created and added to both *Ptolemy II* and D-SPADES.

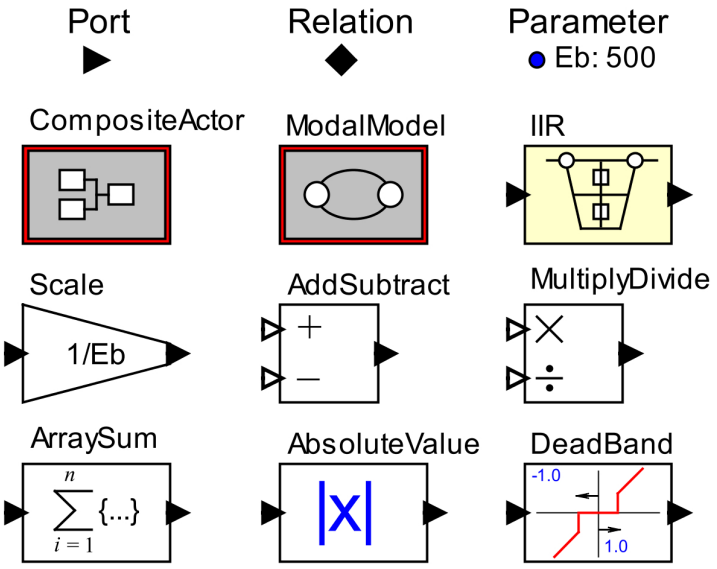
- **Port:** is represented by a triangular symbol. A filled triangle represents a "single port", i.e., a port that receives a link from only one relation. An empty triangle represents

Figure 35 – Actors currently implemented in D-SPADES components library.



Source: The author.

Figure 36 – Some of the graphical elements used within *Ptolemy II* and EMSML.



Source: Neis *et al.* (2023).

a “multiport”, which handle multiple independent channels, i.e., links from different relations.

- **Relation:** implements the mediator pattern, as mentioned above.
- **Parameter:** is a named variable, whose value is specified by an expression which can

refer to constants and even other parameters.

- **Composite Actor:** is an essential block for the creation of hierarchical models. These are models that contain components that are themselves models. The icon shown in Figure 36 is the default appearance of a Composite Actor, however it may also have a custom icon individually attributed.
- **Modal Model:** is a special block used in the creation of models composed of states and transitions, like Finite State Machines. They may also have a customized icon attributed individually.
- **IIR:** implements a parameterized Infinite Impulse Response transfer function. The numerator and denominator coefficients of the transfer function are set by double-clicking on the block and adjusting the desired values.
- **Scale:** produces an output value that is equal to a scaled version of the input, according to a configurable scaling factor.
- **Add Subtract:** implements the functions of adding and subtracting. Its two input ports are "multiports", i.e.: multiple inputs can be connected to the same port. Inputs connected to "+" are added, and inputs connected to "-" are subtracted.
- **Multiply Divide:** implements the functions of multiplication and division, similarly to the "Add Subtract" above.
- **Array Sum:** computes the sum of the elements in an input array, resulting in a scalar output.
- **Absolute Value:** computes the absolute value of the input.
- **Dead Band:** produce an output value that is equal to the input if the input is outside a configurable range, or zero if the input is inside the range.

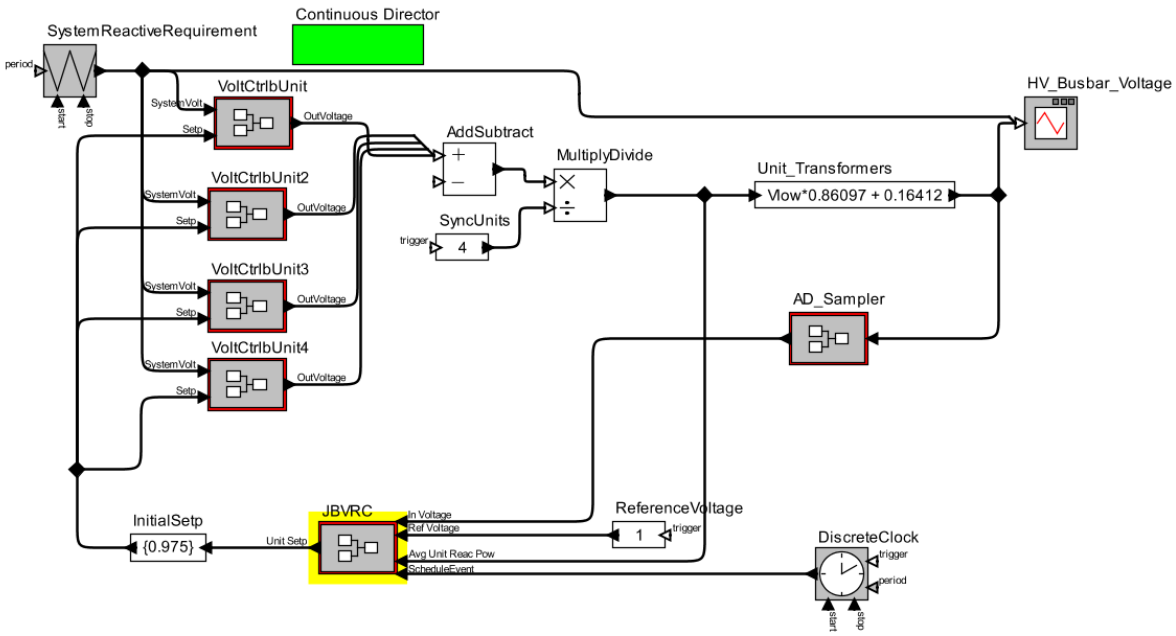
5.2.3 Modeling Language Validation

The metamodel shown in Figure 33 can be validated through the instantiation of its classes. We perform this by creating a model using the *Vergil* editor and verifying the mapping of objects in this model to classes from the metamodel. Later, the model can also be converted

to XMI format and loaded into EMF for further validation: in case the model fails to load into EMF, adjustments need to be made to the metamodel.

The simplified power plant joint voltage and reactive power controller shown in Figure 34(b) can be used for validation. This is actually an oversimplified model of the JBVRC application, shown here for informational purposes. This diagram is a rendition of the same model whose excerpt is shown in Listing B.1[3]. The JBVRC controller shown in Figure 34(b) is also a valid MoML model, so it can be instantiated and used as part of a *Ptolemy II* simulation, as shown in Figure 37. Notice that the entity outlined in yellow (named JBVRC) contains the model shown in Figure 34(b). On the other hand, the diagram in Figure 37 does not necessarily describe a valid EMSML model, since it represents the simulation model of the whole physical process (the environment model), not just the controller we are designing in EMSML (the application model). The hierarchical composition of the actors that compose the CPS model shown in Figure 37 can be schematically represented by the diagram depicted in Figure 38. The model contains, besides other CPS components, the JBVRC controller block to which we must apply the M2T transformation. Blocks drawn with rounded corners correspond to atomic actors, while square corners correspond to composite actors.

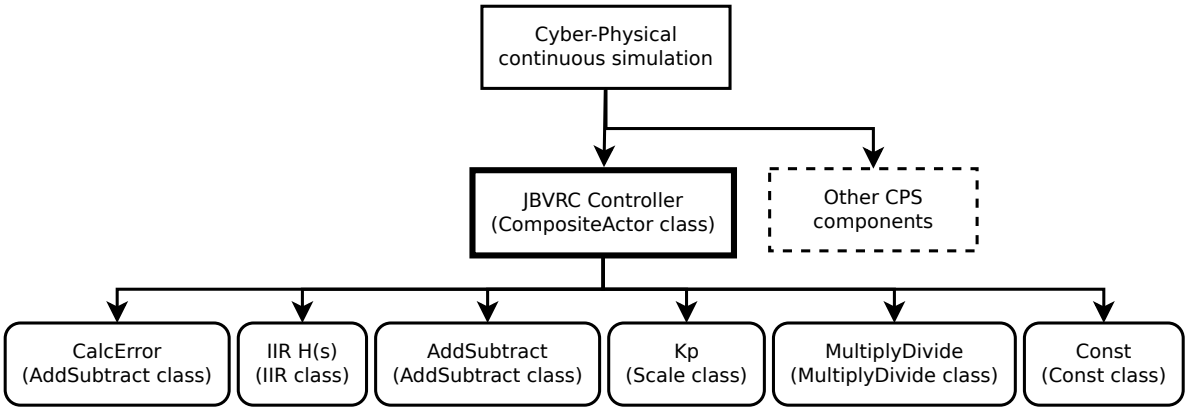
Figure 37 – The JBVRC controller as part of a larger Ptolemy II simulation.



Source: The author.

[3] The *Vergil* editor also serializes the MoML models to an XML format, however, the XML file exported is slightly different from the XMI file needed by D-SPADES, thus a conversion is required, as discussed in Sec. 5.3.1.

Figure 38 – Levels of composition of the toy JBVRC model.



Source: The author.

The correspondence of the instances from Figure 34(b) with the EMSML metamodel classes can be established as follows:

- The “JBVRC” actor is an instance of the “entity” class; notice that it contains other objects, like the “SDF Director” and even other instances of the class “entity”, like the “IIR H(s)” transfer function.
- The input/output ports, like “Ref Voltage” and “Unit Setp” are instances of the class “port”.
- The lines connecting ports and entities are instances of the class “link”.
- For every connection between two or more ports, an instance of the class “relation” exists. Such instances are not necessarily shown explicitly in the *Vergil* editor rendition of the model (Figure 34(b)) but are part of the concrete syntax. Examples of objects of this class are shown in the excerpt from the XMI file shown in Listing B.1 (lines 40–42, for instance).

5.3 MODEL TRANSFORMATIONS

As pointed out by Brambilla *et al.* (2017), the main goal of a model-driven approach to software engineering is to *get a running system out of its models*. In practical cases, like the modeling of EMS application, the “thing being modeled” – the EMS function – needs to be integrated with the base SCADA platform and other legacy applications. The most direct approach to achieve this integration is to produce program source code in a 3GL for the SCADA platform through an M2T transformation. Therefore we propose mapping the actor-oriented models described in EMSML language into an OO (or procedural) programming model through

Model-to-Text transformations. In the Itaipu case study, a significant portion of the legacy systems are developed in C/C++ and Fortran, therefore we choose these languages as targets. We propose a template-based approach to code generation using an M2T transformation language. The template-based approach has several advantages when compared to implementing generators from scratch since it tends to facilitate future changes in the SCADA platform or target programming languages. It is important to highlight that the proposed code generation approach through M2T transformations is not limited to the above-mentioned target languages. Other languages are supported, provided that mapping rules can be established for those languages.

5.3.1 Model Conversion

Prior to the actual M2T transformation, the *Ptolemy II* XML file (Figure 32, page 96 - step 1) is converted into XMI format (step 2). This conversion is performed by a small program, written specifically to manipulate XML files.

5.3.2 Mapping the AO Model into a Sequential Programming Model

The mapping of an AO model described in EMSML into 3GL programming languages is performed through a template-based Model-to-Text transformation, as represented in Figure 32 (page 96 - step 2). The transformation itself is defined using Acceleo Query Language (AQL)[4]. AQL, sometimes called “MTL”, is an implementation of the MOF Model to Text Transformation Language (MOFM2T) specification defined by the OMG[5], and is supported by the Acceleo tool, which is further discussed in Section 5.4.2.

Listing 5.1 shows an excerpt of an MTL file that processes EMSML models – for instance: the JBVRC controller example from Figure 34(b) – and produces C++ source code files as output, as shown in Listing B.3 from Appendix B. The module imports the EMSML metamodel definition (line 1), for which the transformation templates are defined, thus the structure and the properties of the models are known. The AQL template navigates the model and queries the data needed for code generation. Templates are defined for a specific metamodel class, e.g., the *entity* class from Figure 33 so that they generate text as output, which can be directed into files. Such text can be either plainly typed into the template, like the C++ “#include”

[4] https://www.eclipse.org/acceleo/
[5] https://www.omg.org/spec/MOFM2T/1.0/About-MOFM2T/

directives (e.g., line 15), or the result of the evaluation of *tags*, which are statements specified inside the "[.../]" markers (e.g., lines 6-10).

Listing 5.1 – Excerpt of Acceleo MTL template file.

```
1 [module generateComposite('http://www.example.org/EMSML')]
2 [template public generateComposite(anEntity : entity)  ? (self.name = 'JBVRC') ]
3 [comment @main/]
4
5 [comment Generate the header file for the model /]
6 [file (anEntity.name.replaceAll(' ', '_').concat('.h'), false, 'UTF-8')]
7 [anEntity.genCopyrightHeader()/]
8 [anEntity.genClassIncludes()/]
9 [anEntity.genForwardDeclarations()/]
10 [anEntity.genCompositesDeclarations()/]
11 [/file]
12 [comment Generate the implementation (cpp) file for the model /]
13 [file (anEntity.name.replaceAll(' ', '_').concat('.cpp'), false, 'UTF-8')]
14 [anEntity.genCopyrightHeader()/]
15 #include "[anEntity.name.replaceAll(' ', '_').concat('.h')/]"
16 [anEntity.genClassImlementation()/]
17 [/file]
18 [/template]
19
20 [** * This template generates the classes forward declarations*/]
21 [template private genForwardDeclarations(anEntity : entity)]
22 class [anEntity.name.replaceAll(' ', '_')/];
23 [for (anEntityIter : entity | anEntity.entity)]
24 [if anEntityIter.class = 'ptolemy.actor.TypedCompositeActor']
25 [anEntityIter.genForwardDeclarations()/]
26 [/if]
27 [/for]
28 [/template]
29
30 [** * This template generates the header file contents with classes declarations*/]
31 [template private genCompositesDeclarations(anEntity : entity)]
32
33 class [anEntity.name.replaceAll(' ', '_')/] : public CompositeEntity {
34 public:
35     [anEntity.name.replaceAll(' ', '_')/](std::string sEntityName);
36     ~[anEntity.name.replaceAll(' ', '_')/]();
37     bool initialize();
38
39 private:
40     [anEntity.declareComposingEntities()/]
41     [anEntity.declareRelations()/]
42 };
43
44 [for (anEntityIter : entity | anEntity.entity)]
```

```
45 [if anEntityIter.class = 'ptolemy.actor.TypedCompositeActor']
46 [anEntityIter.genCompositesDeclarations()/]
47 [/if]
48 [/for]
49 [/template]
```

Additionally, we have developed and employed M2T templates that can process FSM models and produce procedural source code in the Fortran language. Such templates were employed in the production of modules implementing new functionalities for legacy applications, which will be further discussed in Section 6.3 (page 139).

5.3.2.1 Mapping SDF Models Into OO Programming Model

As a mapping strategy from an AO SDF model to an OO programming language (C++) we propose the following:

1. The model must be built to have one specific composite entity containing the whole desired controller model, which we arbitrarily called “root entity”. The name of this composite entity is passed as an argument to the M2T template, as shown in line 2 of Listing 5.1 (self.name = ’JBVRC’, in this case). For instance, the model illustrated in Figure 37 can be processed by the template by specifying the “JBVRC” as the root entity. This entity corresponds to the block highlighted with thicker line width in Figure 38, while all other entities in the model (other CPS components) are ignored. Source code will be generated for this composite entity (and everything it contains), while other entities like the “AD_Sampler” will be ignored.

2. Every composite contained in the root entity (itself included) is mapped to an entity container C++ class, which in turn is derived from the composite pattern from OO modeling (GAMMA *et al.*, 1995). Instances of these classes are responsible for orchestrating the execution of their contained entities. The template is called recursively for each composite entity contained in the design.

3. The contained entities – the actors and their relations/links – are mapped to form the corresponding model’s graph structure at every containing level.

4. The generated class implementing the root entity must then be instantiated and invoked from within an executable program (e.g. the C/C++ main() procedure).

For instance, the controller previously shown in Figure 34 (page 101) should roughly map to the following structure in an OO programming model:

- One container class (root entity) that interfaces with the SCADA platform. This class' instance reads the following inputs from the SCADA platform: the reference (desired) voltage at the 500 kV busbar; the current voltage at the 500 kV busbar; the average reactive power generation among the generating units under control; the scheduling (periodic) event. As output, it produces the individual generating unit's setpoint voltage.

- For every composite actor in the model, a class declaration and implementation of the corresponding graph at its level in the hierarchy.

- For every non-composite actor in the model, the necessary OO code for instantiating and initializing the corresponding class from the components library, for instance: the IIR transfer function.

- The necessary code for instantiating the relations and establishing the links among entities and relations.

Regarding the execution strategy, the computational differences between the SDF and the sequential execution model supported by current computers have to be accounted for. SDF is inherently parallel, therefore actors perform their computation as soon as sufficient data is available at each of their input ports, regardless of time and independently of other actors in the model. To map this MoC into a sequential model, we adopt a simple strategy: actors are executed sequentially, one at a time (single-threaded execution). The order of actor execution is determined by a scheduling algorithm. D-SPADES uses a simplified version of the scheduling algorithm described by Lee and Messerschmitt (1987). This algorithm can be executed either offline, at build/compile time, or at the executable initialization time (before the program's main loop is invoked). D-SPADES currently runs the scheduling algorithm at program initialization time, but it can be easily moved to the build/compile time.

5.3.2.2 Mapping AO Models Into Structured Programming Model

We have also proposed mapping of AO models into structured programming models, particularly intended for models containing FSMs. The target model consists of Fortran sub-

routines, in order to implement a real-world application needed at the Itaipu power plant. This application is described in Section 6.3.

The models subject to this mapping, as well as the source code generated, must follow specific conventions:

- The "root entity" to be processed by the M2T template must be a "modal model", i.e.: a composite entity containing an FSM. This FSM can be hierarchical, i.e.: one or more of its states may contain refinements, which are instances of FSMs associated with a particular state of the container FSM.
- For each FSM instance in the hierarchical composition, a *Controller* sub-routine is generated in 3GL. This includes the root "modal model" and any other FSM contained in the hierarchical structure. The naming convention for these sub-routines is:
  - *ModalModelName*_Controller() for the root modal model.
  - *StateName*_Controller() for each refinement contained in the FSM;
- The input ports declared in the root "modal model" are mapped as input parameters to each of the controller sub-routines. All the contained FSM controllers receive the same set of input parameters, so they all "see" the same input ports from the model, mimicking the behavior of FSMs in the *Ptolemy II* environment.
- The output ports of the model are mapped to subroutine calls. Therefore, each output action associated with a state transition results in a sub-routine call. These output sub-routines can perform any arbitrary actions needed (for instance, issuing an alarm), and are implemented by hand.
- The model execution is orchestrated by the topmost sub-routine (*ModalModelName*_Controller()). Transitions (guard expressions) are first evaluated at the current level of composition in the FSM. If no transition has been triggered, and the current state contains a refinement, the sub-routine corresponding to that refinement is called. The M2T template relies on the support for recursion in the Acceleo tool to efficiently implement this convention.

To implement the mapping, a specific M2T template was developed, similar to the one shown in Listing 5.1 but focused on FSM models, as well as a "lightweight" version of the D-SPADES components library, implemented in Fortran. Given these conventions, the D-SPADES

workflow shown in Figure 32 is executed in order to process the models, generate code, and integrate it with the SCADA/EMS.

5.3.3 Closing the Gap Between AO Models and Target Code

Since the models expressed in EMSML – likewise MoML – have no associated semantics, there is a gap between the model specification and the target implementation that needs to be filled in order to produce a functional executable application. We argued that EMSML is an adequate language to express EMS applications at a high level of abstraction, however, the details of the actual processing performed by the language elements, like atomic actors and relations, cannot be completely specified by an EMSML model. D-SPADES approach to fill this gap is to supply an implementation of the missing functionalities, which cannot be specified at the model level, arranged into a library that can be used at compile time to produce the running application. The functionalities of this library are implemented directly in a 3GL. Such a library contains the "building blocks" for the application design modeled in EMSML. It is even possible to map some of the actual implementations of these "building blocks" into external services, provided by specialized hardware or networked software, for instance: the IIR transfer function shown in Figure 34(b), instead of being programmatically implemented by the library, could simply be mapped to an I/O operation on a hardware device or into a remote procedure or web service call.

In comparison, in the *Ptolemy II* environment, it can also be observed that the syntactic structure specified by a MoML model does not convey its semantics. In other words, it carries no specific instructions on how to execute the model. The semantics is orthogonal to the syntax (LEE *et al.*, 2002) and is determined by a model of computation (MoC). The MoC decides when actors perform internal computation, update their internal state, and perform external communication. The implementation of both the MoC and the actor's internal computation is part of the *Ptolemy II* environment and is realized either in Java programming language or by the association of previously existing actors into a composite hierarchical structure. In this case, the building blocks for the design are the actors and MoCs provided by the *Ptolemy II* environment, while the design itself is conveyed by the MoML model specification.

Similarly to *Ptolemy II*, the D-SPADES approach proposes to use a *components library* implementing the building blocks necessary for realizing EMS applications. Like in a platform-based design, D-SPADES raises the level of abstraction in the design process, by hiding details

of the implementation technology. This higher level of abstraction comes at a price: the possible design choices are restricted by the available actors and MoCs in the components library, and some additional overhead is added to the model execution. On the other hand, the advantages of such an approach include, as pointed out by Lee *et al.* (2002): the possibility of improving design productivity through abstraction and reuse. We can also point out that, as a domain-specific approach, the paradigms used are familiar to the domain experts, for instance: discrete transfer functions like the IIR shown in Figure 34 (page 101) can implement lead-lag filters, smoothing filters and digital controller blocks. Several other atomic actors performing useful functions for the application domain can be provided in D-SPADES components library, like for instance: limiters, rate limiters, delays, dead bands, etc. – each one of these implemented by specializing the "Entity" class from EMSML metamodel. Familiar pictorial representations can be used in the graphical editor, for instance, those shown in Figure 35 or the ones used for the generator and speed governor shown in Figure 3 (d) (page 37), thus making the language more user-friendly for the domain specialists.

To illustrate the concept, let's look again at the "JBVRC Controller" model shown in Figure 34. The controller itself is part of a larger CPS, whose model is shown in Figure 37. We propose applying the D-SPADES approach to derive a realization of the "JBVRC Controller", targeted at the Itaipu SCADA platform. We can apply a template-based M2T transformation to this controller model, and reproduce its corresponding tree/graph structure by instantiating objects using an OO programming language. The actual processing performed by each object, and the coordination of method calls (message passing) are implemented by the components library. The components library has, for instance, a parameterized implementation of the discrete Infinite Impulse Response (IIR)[6] transfer function, among many others. The same actors and MoCs from this library can be rearranged in a different design and produce a different functionality, for instance: the JCAP application, briefly described in Section 3.4.3.1.

5.4 TOOL SUPPORT FOR D-SPADES

Currently, we do not know of any single integrated environment capable of supporting both the construction of actor-oriented models and the corresponding model transformation necessary for D-SPADES. Therefore we propose to assemble a hybrid toolchain composed of

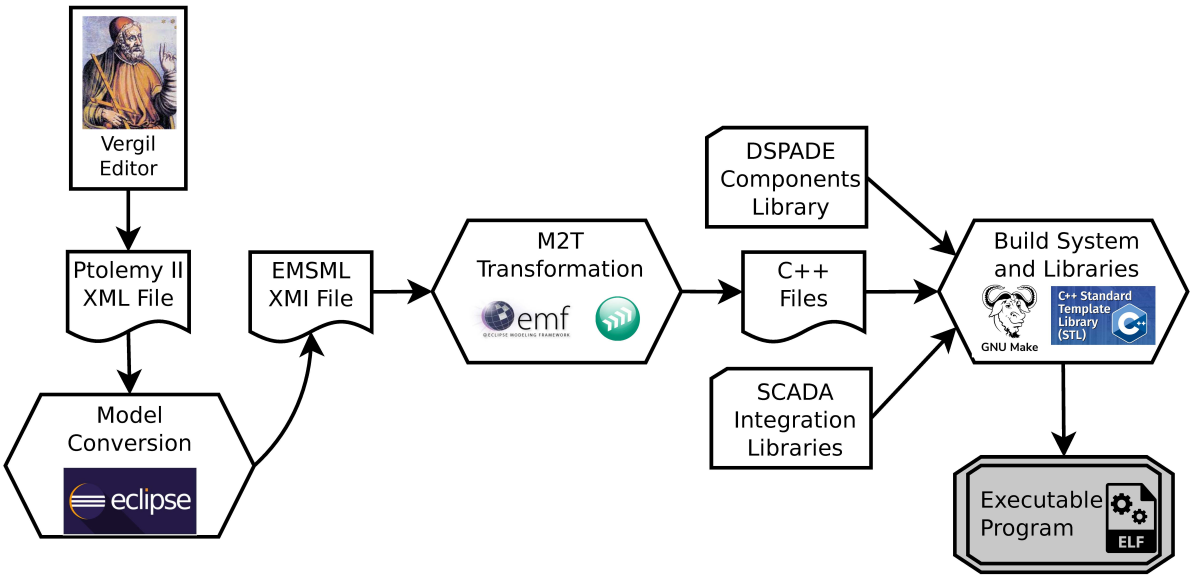
[6] In signal processing, IIR stands for Infinite Impulse Response, and refers to a property of certain linear time-invariant systems that are distinguished by having an impulse response $h(t)$ which does not become exactly zero past a certain point, but continues indefinitely.

community-available, commercial, and in-house-developed tools. This toolchain is composed of:

1. an Actor-oriented Modeling Environment;
2. a Model Processing Framework;
3. the D-SPADES Components Library;
4. SCADA Integration API/libraries;
5. the compiler and build system.

Figure 39 graphically illustrates how these tools are applied in order to produce executable files from EMSML models.

Figure 39 – D-SPADES tool chain.



Source: The author.

These tools are applied along the D-SPADES workflow illustrated in Figure 32 (page 96). The basic execution of this workflow using the proposed tools consists of:

1. build and validate the models using the *Ptolemy II* Actor-oriented Modeling Environment;
2. convert the *Ptolemy II* XML files into XMI format, using a small Java program developed for this purpose;
3. perform the M2T transformation using the Model Processing Tools (Eclipse EMF with Acceleo), along the EMSML metamodel;

4. compile the produced source modules with the target platform's compiler and link with the necessary libraries in order to produce the executable program.

Some of the tools used, like compilers and libraries, can be different from those illustrated in Figure 39, depending on the platform the application is targeted to. In the following sections, a basic description of each of these tools is provided.

5.4.1 Actor-oriented Modeling Environment

Although extensively used for describing models of power systems control applications, the control theory "block diagram / transfer function" paradigm alone is clearly insufficient to represent all the aspects of a software implementation (NEIS *et al.*, 2019). We need support for other useful abstractions, such as finite state machines. FSMs can be used to create hierarchical models known as modal models (FENG *et al.*, 2014). In a modal model, states of an FSM contain sub-models in which each state of the FSM represents a mode of execution, and the mode refinement defines the behavior in that mode.

A modeling environment that supports a rich set of heterogeneous, concurrent modeling paradigms is the *Ptolemy II* from UC-Berkeley, which is an open-source tool. Additionally, *Ptolemy II* can easily perform elaborate simulations of a complete CPS, which is extremely useful for model validation, for instance, by validating the JBVRC controller design shown in Figure 34 (page 101) through simulation of the whole physical process depicted in Figure 37 (page 104). Theoretically, any other actor-oriented modeling tool can be used with D-SPADES, as long as its models can be interchanged with the proposed toolchain.

5.4.2 Model Processing Tools

We performed the metamodeling of EMSML and the M2T transformations using the EMF[7] tools. EMF includes a meta model (Ecore) format for describing models and runtime support for the models including persistence via XMI serialization. For M2T transformation, we have used Acceleo[8]. Acceleo is a template-based technology for creating custom code generators. It supports the automatic generation of any kind of source code from any data source available in EMF format.

[7] https://www.eclipse.org/modeling/emf/
[8] https://www.eclipse.org/acceleo/

5.4.3 Component Libraries

We are not aware of any pre-existing components library that supports all the components needed to implement custom EMS applications through the D-SPADES approach. Therefore we have implemented a C/C++ components library that contains the necessary MOCs and actor processing behavior to be used in Itaipu. Additionally, we have also implemented a limited version of a components library supporting the execution of FSM-based models generated in Fortran.

The components library contains, among other things, the implementation of elements such as those shown in Figure 36. These elements are subject to extensive unit testing in order to ensure that their individual behavior is correct, thus minimizing the risk of introducing defects into the design due to programming errors. For instance, the *AddSubtract* actor is subject to a set of individual and automated tests where different combinations of input data are provided, and the generated outputs are compared to the expected values. After any modifications to the actor source code, the tests are executed again in order to guarantee that it still performs as expected.

5.4.3.1 Integration with Base SCADA

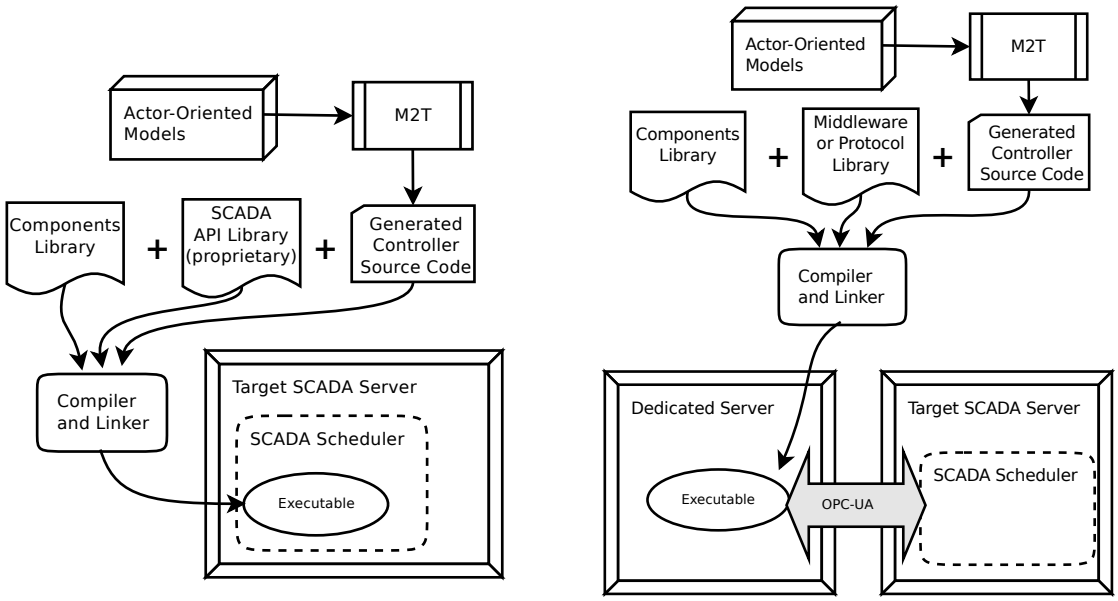
There are at least two possible approaches to integrate applications developed through D-SPADES with commercial SCADA packages:

1. A **tight integration strategy**, in which the EMS application is directly deployed on the existing SCADA servers, using proprietary libraries/APIs for integration. Figure 40(a) schematically represents this type of integration. This approach is feasible only when the SCADA/EMS vendor's API is available, so the integrated application has direct access to internal SCADA data structures and library calls. This simple approach implies less development effort and runtime overhead. A clear disadvantage, though, is that future changes in the vendor's API may break the integration. In a worst-case scenario, a future upgrade to a different product line may no longer grant access to the SCADA API due to contractual arrangements. So far we have utilized this approach for building prototypes and also production applications demonstrating the viability of D-SPADES.

2. A **loose integration** strategy, in which the EMS application is deployed on a dedicated server, and integrates to the base SCADA through standardized protocols/middlewares

like OPC-UA, which figures as a prominent interoperability technology, as suggested by our literature review (NEIS *et al.*, 2019). This strategy is represented schematically in Figure 40(b). A significant advantage of having a standardized interoperability technology is that it may require only minimal migration effort in future upgrades of the SCADA package. Support for the standard protocol may be specified in the bidding process for future provisioning of SCADA software. This is the preferred architecture for the long-term adoption of D-SPADES since it allows easier integration with commercial SCADA packages. However, it requires an OPC-UA connector to exist in the components library, so the EMS application can read and write data to the SCADA. It also requires that the SCADA product fully supports reading and writing operations via OPC-UA. We have implemented basic support for OPC-UA in the D-SPADES components library, and demonstrated the feasibility of this architecture, as discussed in Section 6.2.5.

Figure 40 – Integration architectures for applications coupled with SCADA.



(a) Tight Integration Architecture. **(b) Loose Integration Architecture.**

Source: The author.

Table 4 shows a comparison between the tightly and loosely coupled integration architectures, considering the aspects of portability, scalability, and performance.

Table 4 – Comparison of integration architectures.

Aspect	Tightly coupled	Loosely coupled
Portability	Migration to different SCADA product requires extra integration work	**Easier to migrate to different SCADA product, but requires support for the middleware**
Scalability	Depends on vendor-specific proprietary support for scalability	**More scalable (easily distributed among servers)**
Performance	**Virtually faster and more deterministic (less overhead)**	Might include communication overhead and non-determinism due to the network layer

5.5 EXTENDING D-SPADES

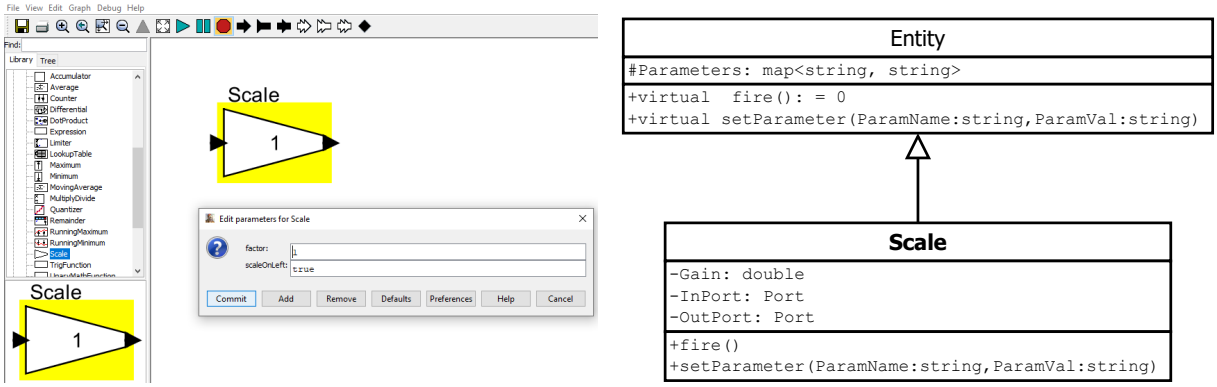
D-SPADES can be extended to support new actor implementations, new MoCs, and different target programming languages. Below we discuss the basic tasks involved in the development of such extensions.

5.5.1 Adding New Actors

The addition of new actors into the D-SPADES library involves, essentially, the development of a specialized class, which implements the interfaces specified by the base D-SPADES "*Entity*" class. If the desired actor already exists in the *Ptolemy II* library, then it can be promptly used by the modelers in the application modeling and validation using the *Vergil* editor. One example of such an actor is the "*Scale*", which basically implements a gain block, controlled by a configurable parameter. Figure 41 illustrates how this existing actor from the *Ptolemy II* "math" library (Figure 41(a)) is implemented into D-SPADES, using a few of its attributes and methods as examples. The "*Entity*" abstract class is specialized by the "*Scale*" class, which implements the "*fire()*" method and overrides the base class' "*setParameter()*" method, in order to set the "*Gain*" value. The "*fire()*" method is responsible for triggering the actor's processing, including the reading of the input token from the "*InPort*", performing the scaling by the "*Gain*" and dispatching the result to the "*OutPort*".

Actors not yet existent in the *Ptolemy II* library can also be added to the D-SPADES library. In this case, however, the new actor implementation (or at least a mock-up) must be also added to the *Ptolemy II* library, so the modelers can add it to their application models using the *Vergil* editor. Apart from that, the process is similar to the one described above. The "*DspadesUAwrite*" actor illustrated in Figure 42, for instance, is implemented by specializing the same "*Entity*" class, as suggested in Figure 42(b). Additionally, a similar implementation of "*DspadesUAwrite*" must be provided (in Java) for the *Ptolemy II* library, using specialization

Figure 41 – Adding existing Ptolemy II actors into D-SPADES.

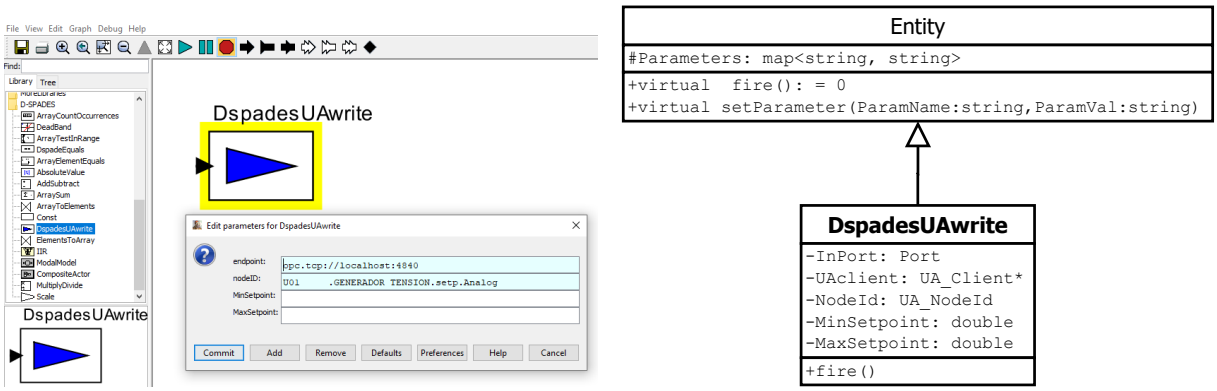


(a) Scale actor in Ptolemy II Library. **(b) Implementing Scale class in D-SPADES.**

Source: The author.

of similar class structure existent in that library. Once that implementation is created, the "*DspadesUAwrite*" actor will be available for use in the *Vergil* editor, as suggested in Figure 42(a). Valid EMSML models instantiating the "*DspadesUAwrite*" actor, once transformed into C++ source code, will instantiate the corresponding implementation from the D-SPADES library.

Figure 42 – Adding new actors into D-SPADES and Ptolemy II.



(a) DspadesUAwrite actor in Ptolemy II Library. **(b) DspadesUAwrite class in D-SPADES.**

Source: The author.

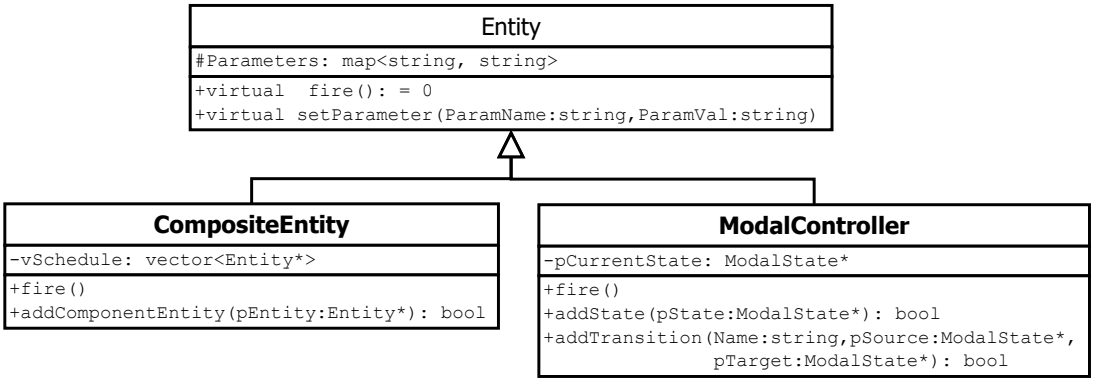
It is worth noticing that an actor implementation may include calls to external libraries, which include networked services or device drivers associated with hardware. This opens the possibility of implementing very specialized actors capable of, for instance: (i) perform complex or high-speed computations using dedicated hardware such as a DSP; (ii) implement adaptive control strategies, or train machine-learning-based models using historical data which can be retrieved from a time series database; (iii) interact with operators through Human-Machine Interfaces. The IEC 62541 (OPC-UA) standard is particularly useful in supporting such

functionality since it allows information to be easily and securely exchanged between diverse platforms from multiple vendors.

5.5.2 Extending the Support for Models of Computation

Support for different MoCs can be added into D-SPADES essentially by modifying the "*CompositeEntity*" class implementation, schematically represented in Figure 43. This class also extends the base "*Entity*", and therefore implements its own "*fire()*" method. In the particular case of the SDF MoC, the "*fire()*" method simply executes the contained actors in the order defined in the scheduling vector, which is pre-determined.

Figure 43 – CompositeEntity and ModalController classes from D-SPADES.



Source: The author.

A different MoC would require a different strategy for executing the contained actors. As an example, consider the Discrete Event MoC, in which actors communicate through events ordered in a timeline (LEE, 2014). In such a MoC, the order of execution of the actors would be determined by an event queue. Therefore the "*CompositeEntity*" should be modified in order to support such event queue, and choose the appropriate execution strategy according to the specified "*Director*" class contained in the model (see the EMSML metamodel from Figure 33, page 100). One particular case, which is already supported by D-SPADES is the FSM MoC: it is currently implemented as a separate specialization of the "*Entity*" class, called "*ModalController*", illustrated in Figure 43.

5.5.3 Different Programming Languages

Support for different target programming languages in D-SPADES can be achieved essentially by writing new M2T transformation templates, such as the one shown in Listing

5.1 (page 107). The D-SPADES components library could be linked to compiled objects built from different languages, as long as binary compatibility is supported by the compilers being used, ie: the binary code generated from the languages is compatible at the Application Binary Interface (ABI) level. For instance, a template for generating code in the "C" language could be created, and the produced "C" module could be compiled and linked to the existing "C++" binaries of the components library. Of course, that would impose limitations, particularly on the aspects of object orientation, which are not supported in pure "C". Another option would be to produce a full re-implementation of both the components library and templates, although that would represent a significant effort. In Section 6.3 we show an application where we actually decided to implement an alternate version of the components library (although with limited functionalities) to support a legacy system implemented in Fortran. This example reinforces the idea that D-SPADES can be independent of the target platform and target programming language.

5.6 REMARKABLE FEATURES OF D-SPADES

One of the advantages of D-SPADES, in comparison with traditional development approaches, is the possibility of incrementally designing and validating the controller against the model of the physical process since the early stages of development. This incremental design and validation are supported by using hybrid simulation and design environments such as the *Ptolemy II*. The physical process can be modeled with the necessary level of detail, so it adequately represents the interactions between the controller being designed and the power system. The domain specialist can use the same environment to specify the design, test its performance using the physical process model, and iteratively perform changes on the controller model, thus increasing the chances of producing an adequate design. From this point on, the automatic model transformation techniques take over and generates high-quality source code, which in turn results in executable software virtually free of design flaws and bugs commonly introduced by manual programming.

It is worth highlighting that, although the transformations discussed have not been mathematically/formally proven, the mentioned software quality might be obtained if the mapping between the input model and the output target language is syntactically and semantically correct. In other words, this can be achieved if the mapping between model elements and the target language constructs and services of the target execution platform were extensively tested and

verified. In this sense, the benefits and the gains of D-SPADES are obtained over time through the further reuse of model elements, transformation rules, simulation results, and other previously created high-level and platform- and application-independent design artifacts.

Another sensible advantage of D-SPADES is the support for multiple computational models, like SDF and FSM. Commercially available design frameworks usually define fixed MoC: Matlab/Simulink, for instance, use a continuous time MoC, with discrete time treated as a special case (ZHOU *et al.*, 2007)[9]. Additionally, D-SPADES is extensible: new actors and even new MoCs can be developed and added to the components library, while commercial tools are usually closed to such additions.

In terms of code generation, it can be argued that existing AO design environments already export source code from models, since that is a feature known to exist in traditional tools like Matlab/Simulink and Ptolemy II itself. Such code generators, however, have known limitations, like the intermingling of static and dynamic code, non-graspable output structure on the generator, and production of unnecessarily large amounts of code (BRAMBILLA *et al.*, 2017). D-SPADES on the other hand adopts a template-based approach to code generation, which is designed to deal with the above limitations. Another drawback of traditional code generators is that each modeling element usually needs a corresponding code generator. Ptolemy II, for instance, uses helper-based code generators (ZHOU *et al.*, 2007), responsible for generating target code for a given actor in each specific target language. In other words, it requires a new code generator to be developed for each new actor added to the environment, requiring multiple helpers to support multiple target languages. Moreover, code generators yield little control over the code generation process, for instance: the language and constructions used are pre-determined by the code generator implementation. The template-based approach of D-SPADES helps in dealing with these limitations. In D-SPADES, the addition of new actors to the environment usually does not require changes to the M2T template.

Paz and Boussaidi (2020) emphasize another important limitation of closed technologies, such as Matlab/Simulink:

> MathWorks Simulink and Stateflow comprise one of the most widely used MDE frameworks for developing embedded and safety-critical systems. These MathWorks' technologies and their supporting tooling are focused around satisfying three specific MDE needs: design modeling, simulation and code generation. The outlook, however, is limited since direct extensions to address

[9] It could be argued, though, that Stateflow extends Simulink with a state-like formalism variant of Harel's Statecharts. Stateflow enables the representation of control functions that are dependent on a combination of past and present logical conditions (PAZ; BOUSSAIDI, 2020).

> other additional MDE needs are not possible due to MathWorks' closed nature (PAZ; BOUSSAIDI, 2020), p. 1.

With D-SPADES, the developer has control over the M2T template, which may not be the case with proprietary technologies, where the code generating engine is often seen as a "black box", *which are not easy to trust and produce sub-optimal code* as pointed out by Harrand *et al.* (2016). Additionally, D-SPADES easily accommodates new templates developed for different languages/platforms. This advantage has already been explored during the development of this work, which has demonstrated functional C++ and Fortran templates.

5.7 CHAPTER SUMMARY

In this chapter, we proposed an approach to EMS applications development: the Domain-Specific Power Applications Development Environment and Strategies – D-SPADES. The main components of D-SPADES were described, including:

- The EMSML modeling language, which can be used to describe actor-oriented models;
- The transformations applied on the models in order to produce executable artifacts;
- The necessary tool support for the approach, including modeling tools, transformation languages, components library, and SCADA integration API;
- And the basic software process that coordinates the execution of the development activities.

D-SPADES is focused on raising the level of abstraction in which EMS applications are modeled, similarly to the concept of platforms discussed in Section 2.3.3 (page 39). Such a platform can improve productivity by leveraging the domain paradigms and promoting the integration of power systems specialists into the software development process, without necessarily requiring computer programming skills. D-SPADES does not imply that 3GL and "traditional" development approaches are no longer necessary: it merely provides a layer of modeling that allows the power system specialist to contribute to the EMS software design without necessarily having concerns about the layers of modeling below (LEE, 2018). The roles of individuals working at different layers are complementary and integrated, for instance: developers of 3GL code produce reusable components (actors) that can be used by the power systems specialists at the layer above.

6 APPLYING D-SPADES TO CONSTRUCT FUNCTIONAL APPLICATIONS

In this chapter, we demonstrate how running applications can be generated from high-level AO models using D-SPADES. First, we apply D-SPADES to build a simple toy application that takes as input some process variables, and issues control actions according to a transfer function, for informational purposes. Following, we build and evaluate a proof-of-concept application modeled according to the specifications of a real-world voltage and reactive power controller from the Itaipu Power Plant. This application is compared with the legacy executable in terms of functional and computational performance, showing satisfactory results. Finally, we apply D-SPADES to build a production application successfully deployed at the Itaipu Power Plant. This application, called ERG60, is responsible for supplementary protective actions associated with the transmission system that connects Itaipu to the Brazilian interconnected power system. Thus we demonstrate that D-SPADES is a viable approach for real-world, mission-critical applications.

6.1 RUNNING EXAMPLE: SIMPLE CLOSED-LOOP CONTROLLER

This illustrative example demonstrates how to apply D-SPADES for building and transforming a *Ptolemy II* model into valid C++ source code. This source code can be compiled and linked with the D-SPADES library to produce functional executable programs. The output of these programs can be compared with the results of *Ptolemy II* simulations of the same model to demonstrate the correctness of the produced code.

The requirements for this hypothetical voltage controller application are the following:

1. it shall receive as inputs the following variables, expressed in pu: the measured busbar voltage, the average reactive power produced by the plant's generating units, and the desired busbar voltage (plant's setpoint).

2. it shall compute the individualized generating unit's setpoints (pu), using discrete transfer functions and simple add/subtract operations, for every set of input variables received.

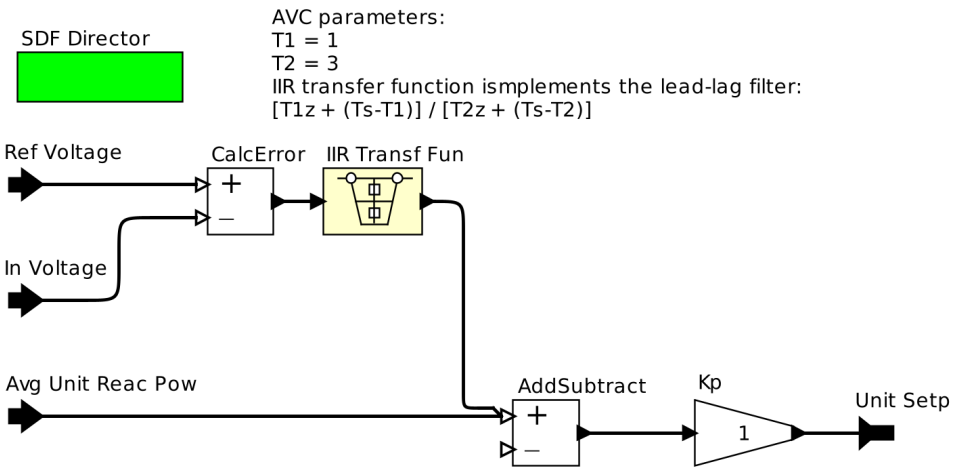
Figure 44 shows one possible realization of such a hypothetical controller, which is a slightly modified version of the JBVRC model from Figure 34 (page 101). This model is placed into a simulation environment shown in Figure 45, composed only of a sequence of input

values and a display output. In this simulation, both the average reactive power and the controller reference voltage are kept constant at 1.0 pu, while the following sequence of input voltage measurements is simulated: {1, 1, 1, 1, 1.01, 1.01, 1.01, 1.01, 1.01, 1.01}.

The JBVRC controller shown in Figure 45 is processed using a M2T template similar to the one shown in Listing 5.1 (page 107) to produce source code. In this case, the source code consists essentially of the "JBVRC" C++ class, composed of a header file (JBVRC.h) and an implementation file (JBVRC.cpp). These files are shown in Listings B.2 and B.3 from Appendix B. The automatically produced JBVRC source code is compiled and linked with the D-SPADES components library, which implements the building blocks of the model, such as the "Port" and the "AddSubtract" components. Finally, in order to execute this toy JBVRC on the operating system's prompt, a simple "main" C++ module (handwritten) invokes the JBVRC for every argument passed through the command line. This main module is also shown in Listing B.4, from Appendix B.

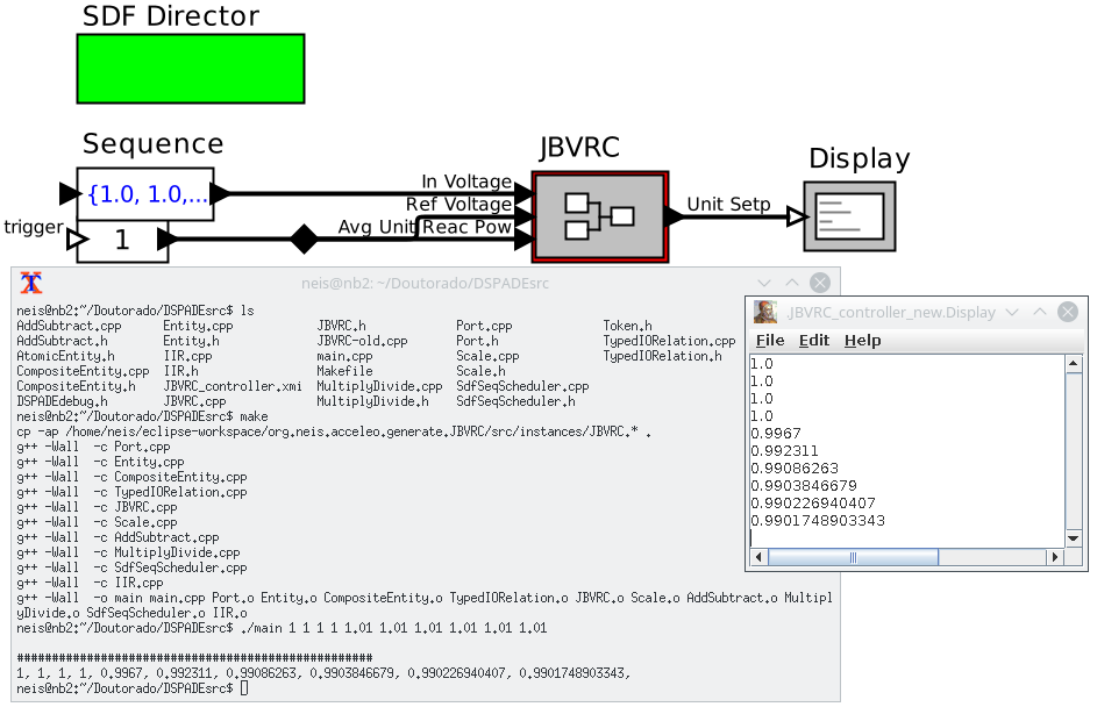
Figure 45 shows results obtained from both the *Ptolemy II* display and the system terminal running the "main" executable. The output is coincident in both environments and consists of the following sequence of setpoints: {1, 1, 1, 1, 0.9967, 0.992311, 0.99086263, 0.9903846679, 0.990226940407, 0.9901748903343}.

Figure 44 – Contents of the composite JBVRC entity implemented a simplified controller.



Source: The author.

Figure 45 – Comparative results of a sequence of input voltages in both Ptolemy II and D-SPADES.



Source: The author.

6.2 CASE STUDY 1: JBVRC APPLICATION

In order to demonstrate the applicability of D-SPADES to real-world scenarios, we replicated an existent EMS application, called *Automatic Voltage Control* (AVC), from the Itaipu hydroelectric power plant, described in Section 3.4.3.2 (page 70). The AVC performs functions related to secondary voltage regulation at the power plant level, thus belonging to the category of "Centralized Control" according to the IEE Std. 1249 (IEC/IEEE, 2013), discussed in Section 3.3. It actually consists of joint control of the generating units' voltage and reactive power, having two main objectives: (1) maintaining the bus bar voltage approximately constant, according to the scheduled voltage, and (2) maintaining an even distribution of reactive power among the operating generators. It is classified as a continuous control[1], and operates in the timescale of seconds or longer (see Figure 19 from page 59). Such an application is sometimes called *Joint Bus Voltage and Reactive Power Control* (JBVRC) (CALOVIC; JELIC, 1992). From now on we will use the term AVC to refer to the legacy application, and JBVRC to refer to the proof-of-concept application developed through D-SPADES.

The AVC application was developed using traditional methods, directly applying a

[1] The term "continuous" refers to the continuously active nature of the controller, as described in Section 3.3.2 (page 58), not to be confused with the concept of continuous time models.

structured programming language. In that approach, modeling activities were limited to early design and documentation of the desired system's behavior. By choosing this existing and consolidated application, which has been operating for approximately 20 years, we can compare both the functional correctness of the executable produced through D-SPADES, as well as assess the viability of the proposed MDE process in comparison with the conventional process focused on programming languages.

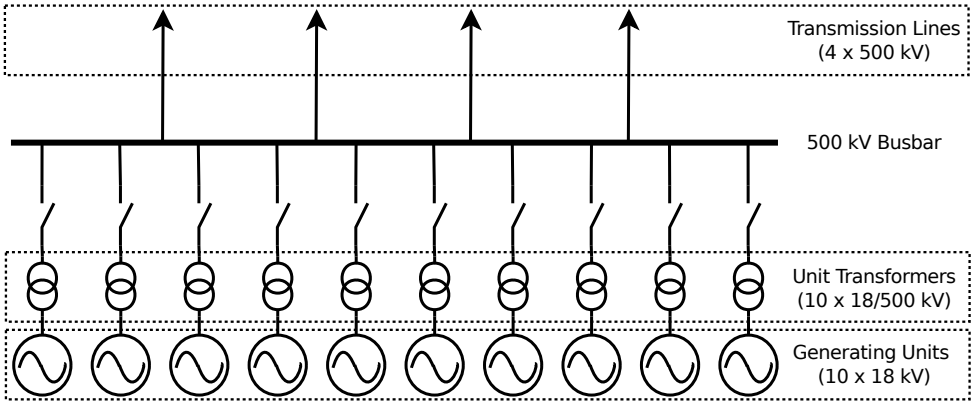
6.2.1 Equipment and Systems Involved

As explained in Section 3.2.2.1, the synchronous generator excitation system follows a reference signal that shall be provided either by the operator or by a supplementary controller. At the Itaipu power plant, the current supplementary controller is the "AVC" function. Figure 46 is a simplified one-line diagram representation of the Itaipu power plant's equipment configuration, for either the 50 Hz or 60 Hz sectors. It schematically represents the 10 generating units, with each one's respective step-up transformer bank, the 500kV busbar, and the four transmission lines. The JBVRC function shall operate by monitoring the voltage at the 500 kV busbar and actuating on each unit's excitation system in order to achieve two main goals:

1. To maintain the voltage at the 500 kV busbar approximately constant, following the established reference.

2. To share evenly the reactive power production among the generating units under its control.

Figure 46 – Simplified one-line diagram of either 50 Hz or 60 Hz sectors in Itaipu.



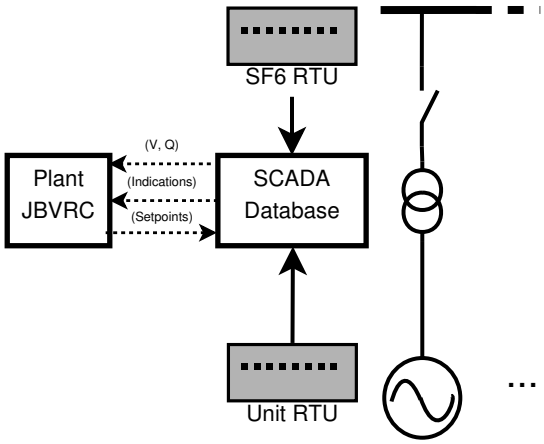
Source: The author.

The JBVRC's interface with the physical process is made through the SCADA platform, as represented schematically in Figure 47. The SCADA collects measurements and indications

from RTUs and or other field devices, and is also capable of issuing control actions over certain plant equipment, for instance: performing switchgear operations and altering equipment operational references. The following data, needed for the JBVRC application, are collected via the power plant's SCADA system:

- voltage at the 500 kV busbars;
- voltages at each generator's terminals;
- reactive power produced by each generator;
- network topology information;
- other relevant indications, like units' synchronization and limiters' status.

Figure 47 – Simplified data acquisition diagram from busbar and generating units.



Source: The author.

6.2.2 Overall Requirements of the JBVRC Application

Some of the main requirements for Itaipu's JBVRC were extracted from current system specifications and are presented below, in an unstructured format. The complete requirement specification can be obtained from existing Itaipu documentation and is not transcribed here since these details are not necessary for this overview.

- JBVRC must receive the 500 kV bus voltage reference from the load dispatcher. This reference is coordinated with the regional System Operators.
- It must support periodically scheduled execution, every two seconds.

- For every execution cycle, it must:
  - fetch and filter raw data from the SCADA telemetered real-time measurements database;
  - calculate the voltage error and determine the modes of operation for generators and the controller;
  - calculate desired unit terminal voltage and issue setpoint control commands;
  - provide error checking of input data to ensure data integrity and consistency, i.e. JBVRC should not perform operations based on invalid or inconsistent input data.

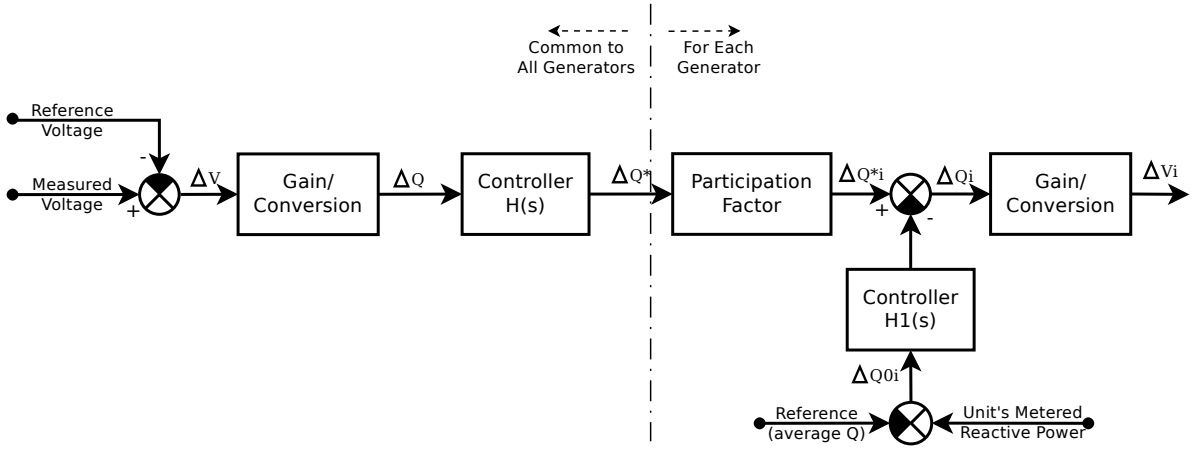
Figure 48 represents a (sketch) model of Itaipu's current AVC application. This model served as input for deriving the actor-oriented EMSML model for the JBVRC. In this sketch it is possible to identify two separate sections: on the left-hand side, a section that is common to all generating units in a sector; this part of the model is responsible for calculating the 500 kV busbar voltage error, applying appropriate gain with conversion factor from voltage error (ΔV) to reactive power error (ΔQ), and the desired controller transfer function $H(s)$, obtaining ($\Delta Q*$). The right-hand part of the model is applicable to every generating unit under JBVRC control: it adjusts the proportion of $\Delta Q*$ that will be shared by each generator (participation factor), obtaining the $\Delta Q*_i$ reactive power error share; calculates each unit's deviation from average produced reactive power (ΔQ_{0i}); applies an inner loop desired transfer function $H_1(s)$ and gain; composes the resulting unit's reactive power error (ΔQ_i), using the individual deviation and the share of the sector's error; and converts the reactive power error back to voltage error (ΔV_i), in the generating unit's base voltage[2]. Other requirements like input data validation and tracking logic are not represented in this diagram and will have to be modeled using other sources of information, like textual descriptions and equations.

6.2.3 Modeling the Physical Process

Once the EMS application requirements are elicited, the next step consists in modeling the physical process it must interact with. The power plant model is composed of 10 generators with their respective step-up transformers, operating in parallel at the high voltage bus bar, rated at

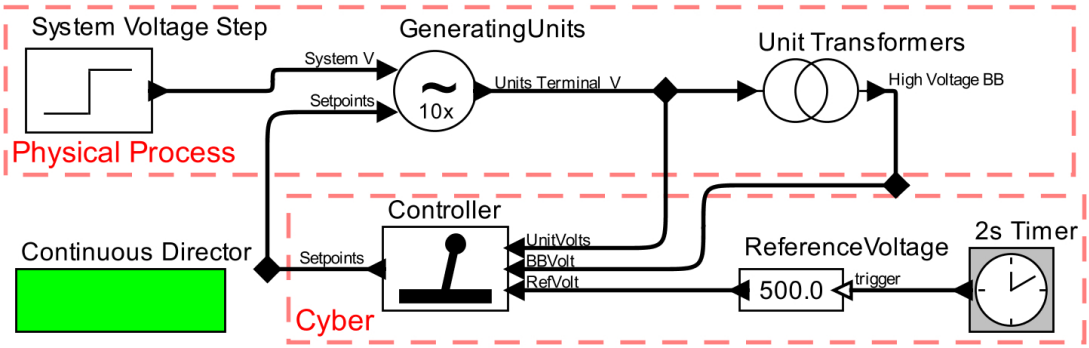
[2] Most of these calculations are performed using the normalized, dimensionless "per-unit system" (pu). Per-unit quantities are converted back to engineering units when the controller interfaces with the power system's equipment, using the appropriate base units of the corresponding equipment. For reference in the per unit system, see (KUNDUR *et al.*, 1994).

Figure 48 – Current AVC reactive power based control diagram.



Source: The author, reproduced from Itaipu's internal documentation.

Figure 49 – Model of power plant and secondary voltage controller.



Source: Neis *et al.* (2023).

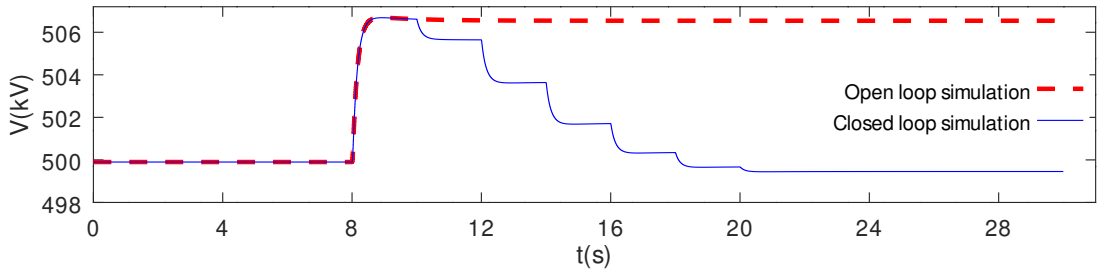
500 kV. Figure 49 shows the higher level of the hierarchical model built using the *Ptolemy II* suite. This level shows the main components of the voltage regulation system: the physical process, including the generating units and step-up transformers; and the associated cyber component responsible for the secondary voltage regulation. The physical process also contains a component modeling the equivalent to a system voltage step variation, resulting from a sudden change in either active or reactive power consumption by the associated power system (generator, load, or capacitor bank shedding, for instance). The green box at the bottom left hand-side indicates to *Ptolemy II* that, overall, this model shall be handled as a continuous-time, although parts of the simulation may execute under different MoC, like the discrete-time computerized voltage controller, which follows the SDF MoC. The hierarchical composition of the physical process, including the generating units and step-up transformers model, is illustrated in Appendix C.

Each generator performs primary voltage regulation by means of its AVR, which has a droop characteristic to ensure proper sharing of reactive power among units connected to the

same bus bar, either directly or through individual step-up transformers. The AVR's droop, in addition to the step-up transformers' regulation effect under varying load conditions, introduces a steady state control error at the high voltage bus bar, hence the need for the secondary voltage controller.

Figure 50 shows a simulation, executed within the *Ptolemy II* environment, of the effect of such a step on the voltage at the power plant's 500 kV bus bar (at 8s), with the secondary voltage controller disabled (dashed curve in red). This simulation roughly mimics the effect of a 350 Mvar capacitor bank being connected at the nearby Foz do Iguaçu substation. In order to counteract this steady state error, supplementary control actions are necessary, such that the output voltage is brought back within an acceptable operating range. These actions, known as *secondary voltage regulation*, are usually determined on a system-wide basis by the regional load dispatcher or system operator authority. Large power plants participate in the secondary voltage regulation by following a voltage or reactive power reference determined by the regional dispatcher. In our example, this reference is directly communicated, on demand, by the system operator to the power plant dispatch room, which manually set the controller's reference. Figure 50 shows, in blue, a simulation of the effect of the same voltage step, with secondary voltage control active, and reference set at 500 kV. Notice that the voltage is gradually brought back to 500 kV by the digital controller, in incremental actions every 2 seconds. This is the system's expected behavior since the secondary voltage controller is always active under normal conditions.

Figure 50 – Ptolemy II Simulation results.



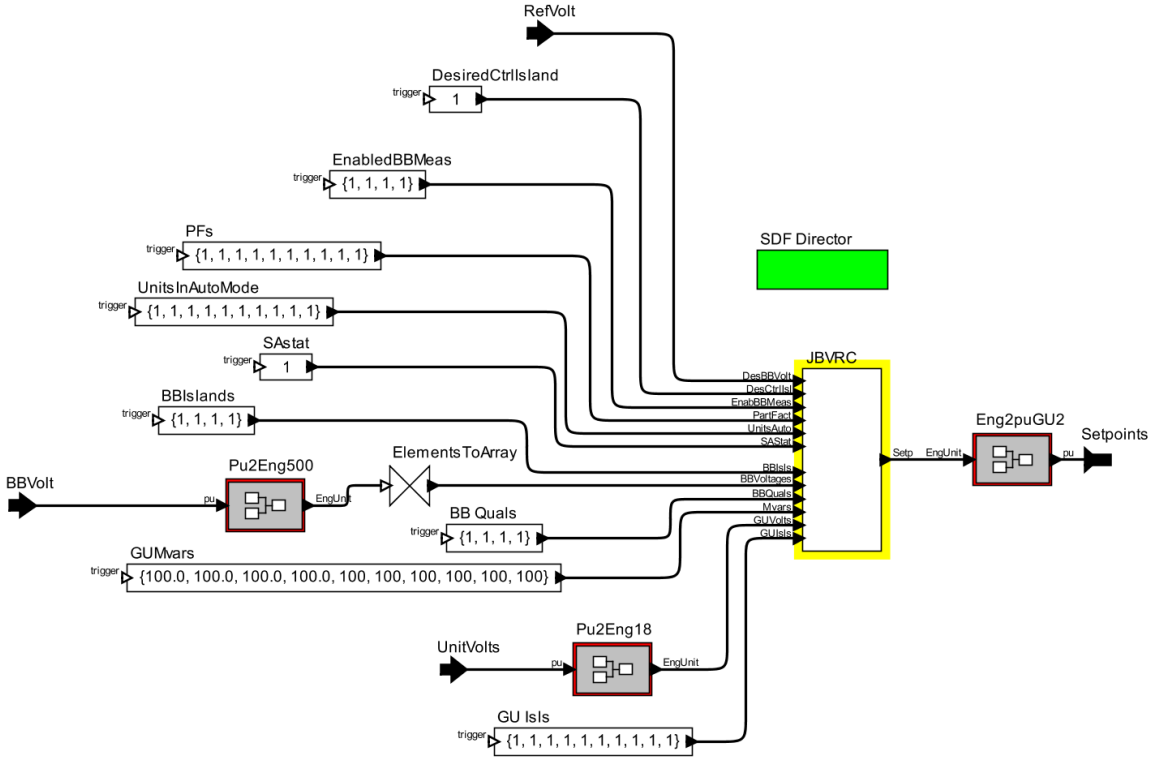
Source: Neis *et al.* (2023).

6.2.4 Modeling the JBVRC Application

The "Controller" block from Figure 49 is a composite actor, executing under SDF MoC, that contains the actual JBVRC block, which in turn implements the power plant level secondary

control actions. The hierarchical composition of the “Controller” block is shown in Figure 51. Since the simulation model shown in Figure 49 is highly simplified, many inputs to the JBVRC block are not modeled, and therefore are “hard-coded” as constant values shown in Figure 51. For instance, the generating units’ reactive power value and its electrical connectivity island are not simulated by the above model, therefore they are kept fixed as “100 Mvar” and “1”, respectively, for simulation purposes. In the real system, however, these inputs are provided by the SCADA system to the JBVRC block.

Figure 51 – Hierarchical composition of the Controller.

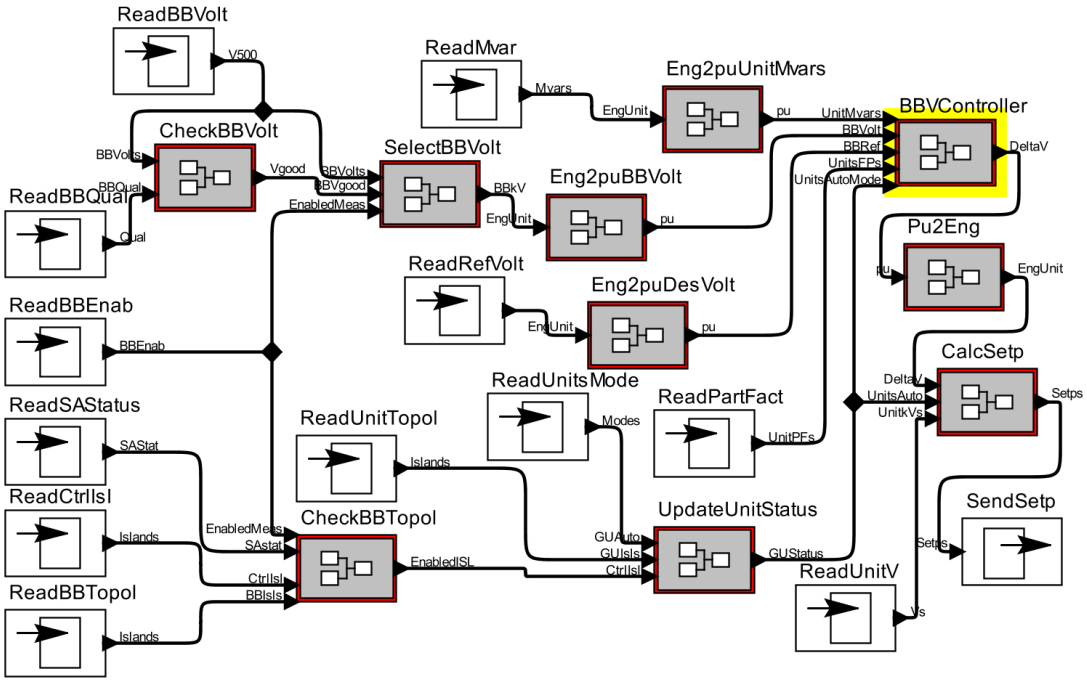


Source: The author.

The JBVRC composite actor highlighted in Figure 51 is hierarchically composed of the model depicted in Figure 52, which performs the actual controller function in the model depicted in Figure 49. This container is the “root” element which is processed by the M2T transformation template, similar to that shown in Listing 5.1 (page 107). The JBVRC actor must perform all the actions required by the application. It will be periodically scheduled by an external entity, which is not represented in the model. Communication with the base SCADA is implemented by specialized actors, named with the “Read” or “Send” prefix in Figure 52.

The voltage controller block emphasized in Figure 52 (*BBVController*) is detailed in Figure 53. This block is responsible for calculating the necessary voltage variation on each generator as a function of the high voltage bus bar error and the reactive power sharing criteria,

Figure 52 – The JBVRC actor model.



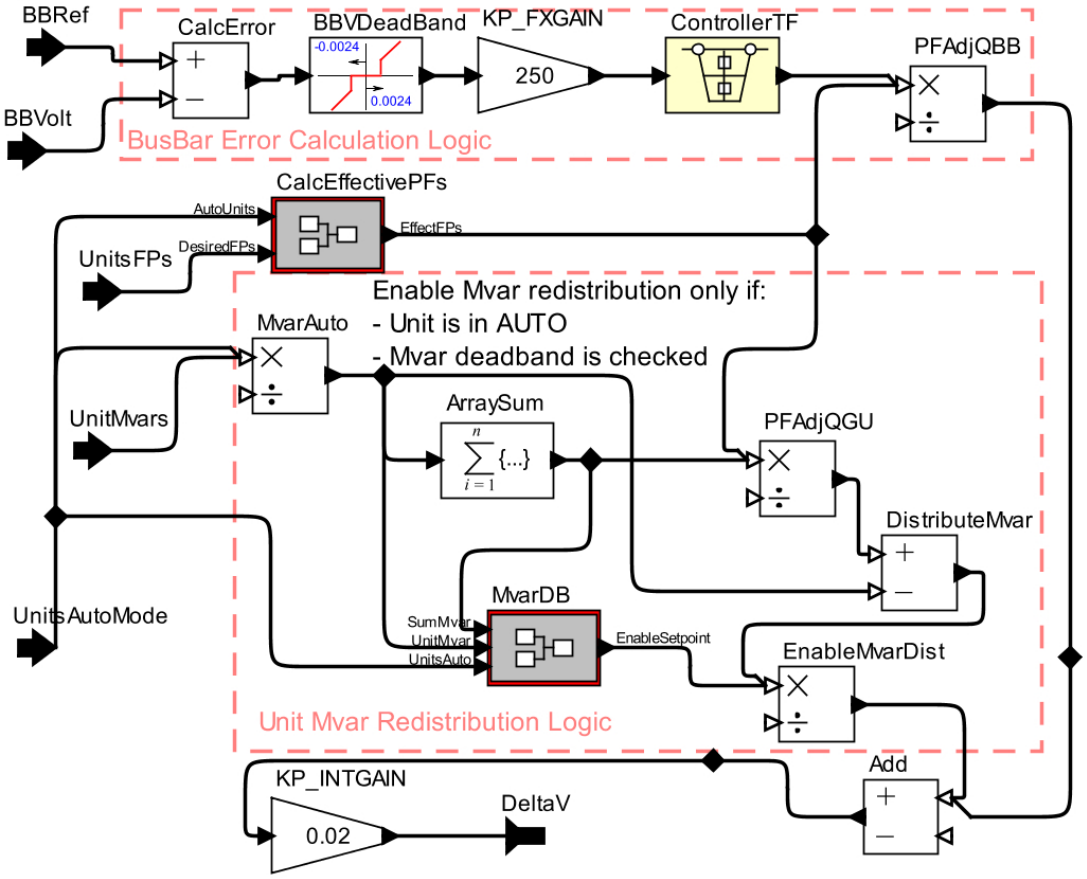
Source: Neis *et al.* (2023).

both weighted according to individually adjustable participation factors attributed to each unit. This model was built based on the stated requirements of the current AVC, and the legacy model described in Section 6.2.2.

The hierarchical composition of the *BBVController* still descends further into other composite actors, for instance, the *MvarDB* block, whose composition is shown in Figure 54. This block implements the Mvar dead band logic on the array of measurements corresponding to each generating unit.

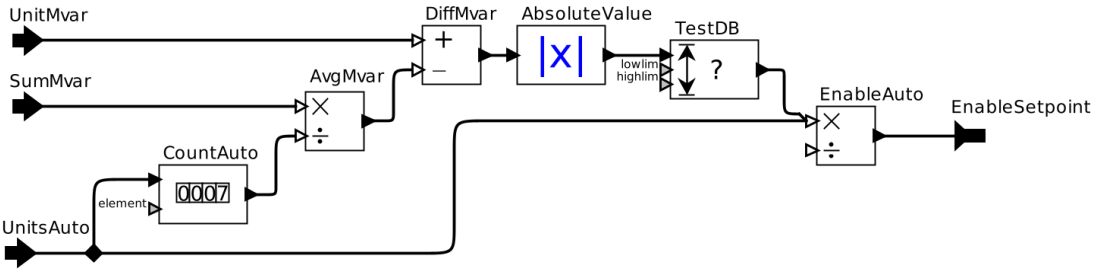
A schematic diagram illustrating the levels of composition corresponding to the entire model from Figure 49 is shown in Figure 55. Blocks drawn with rounded corners correspond to atomic actors, while square corners correspond to composite actors. The root element processed by the M2T template is drawn with a thicked line width. The remaining hierarchical levels correspond respectively to: the *Controller* block, shown in Figure 51, whose composition corresponds to level (3); the *JBVRC* block, shown in Figure 52, whose composition corresponds to level (4); the *BBVController* block, shown in Figure 53, whose composition corresponds to level (5); and finally the *MvarDB* block, shown in Figure 54, whose composition corresponds to level (6). In this diagram, some of the composing blocks are omitted to save space.

Figure 53 – The JBVRC main controller block.



Source: Neis *et al.* (2023).

Figure 54 – The MvarDB block implementing the dead band.

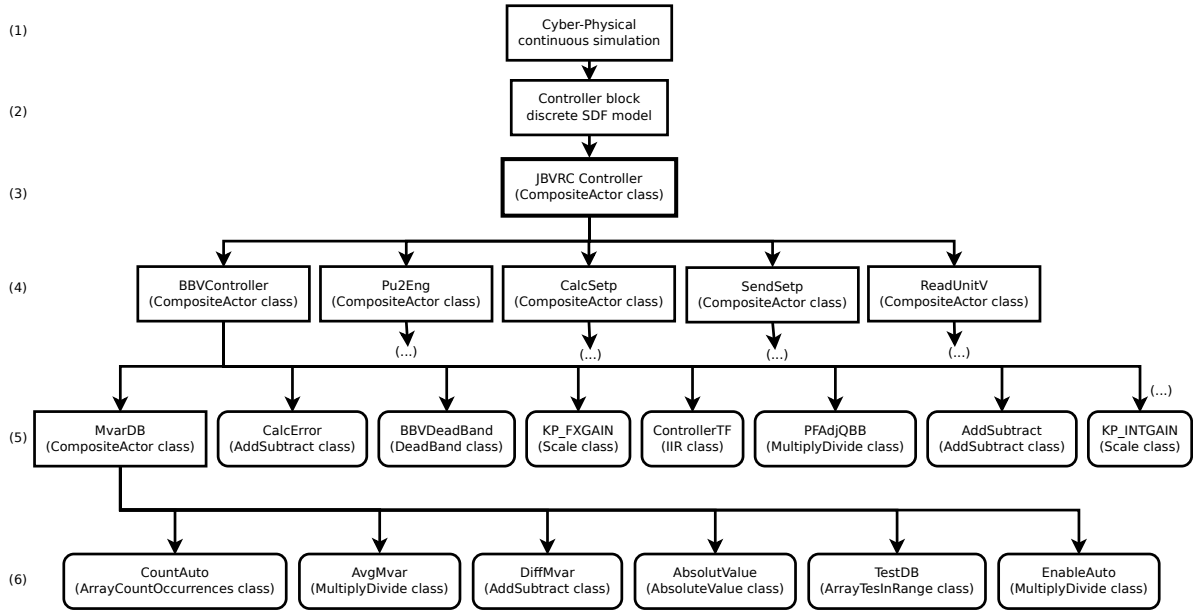


Source: The author.

6.2.5 Integrating JBVRC Into the Base SCADA

We have evaluated the integration of JBVRC to the Itaipu's DTS using both the tight and loose integration strategies described in Section 5.4.3.1 (page 115). In the tight integration strategy, the blocks named with the prefix "Read" and "Send" in Figure 52 are implemented as calls to the native SCADA library API, performing the data access calls needed for both reading variables from and sending setpoint commands to the SCADA layer. In the loose integration strategy, these blocks are implemented using OPC-UA client actors, which were built into the

Figure 55 – Levels of composition of the JBVRC model.



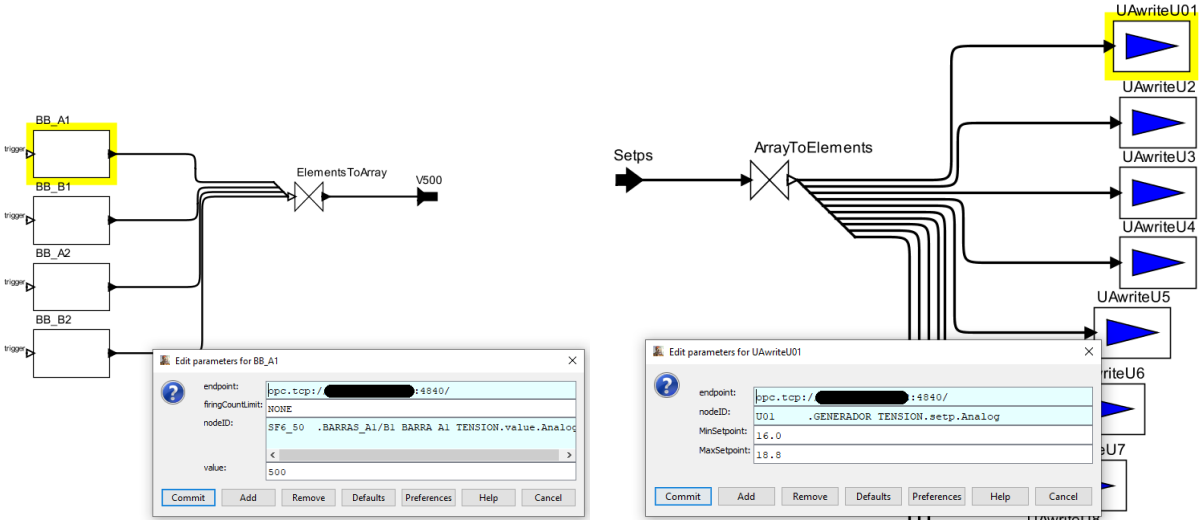
Source: The author.

D-SPADES components library. Figure 56 shows the contents of the "ReadBBVolt" (56(a)) and the "SendSetp" (56(b)) block from Figure 52. In the Vergil editor, by double-clicking on the input/output OPC-UA client actors highlighted in yellow, a pop-up window is shown, so the engineer can configure the communication parameters, such as the server endpoint address and the node ID (tag name) in the server's address space. Such parameters are saved to the *Ptolemy II* XML file and later used in the instantiation of the corresponding D-SPADES objects. The D-SPADES implementation of the OPC-UA client uses the *Open62541*[3] library, which is a certified implementation of the IEC 62541 standard, in conformance with the "Micro Embedded Device Server Profile" of OPC Foundation[4]. We have also developed *Ptolemy II* versions of these actors using the *Eclipse Milo*[5] implementation of the IEC 62541 standard, in Java.

The observed functional performance of the application integrated through these strategies was indistinguishable based on the set of tests executed. We acknowledge that the additional network layer added by the loose integration through OPC-UA, particularly when running JBVRC in a separate server, might negatively influence the performance due to communication latency and possible timing non-determinisms. However, we have not performed a thorough evaluation of these effects in the current stage of this work. On the other hand, the OPC-UA is a shop-floor standard for process control, already proven in many industrial applications

[3] https://www.open62541.org/
[4] https://opcfoundation.org
[5] https://projects.eclipse.org/projects/iot.milo

Figure 56 – JBVRC integration through OPC-UA.



(a) JBVRC reading variables through OPC-UA. **(b) JBVRC sending controls through OPC-UA.**

Source: The author.

worldwide, therefore it should be able to perform satisfactorily with EMS applications.

6.2.6 Results

We have generated the executable software corresponding to the JBVRC controller from Section 6.2.4 and integrated it with Itaipu's DTS. This approach allows a direct performance comparison between the existing AVC and the JBVRC under the same scenario, as described below.

6.2.6.1 Source Code Metrics

We have performed a comparison of software size, in terms of *Source Lines Of Code (SLOC)* metric, of both AVC and JBVRC, as shown in Table 5. Although we acknowledge the SLOC metric may not be an appropriate metric of software functionality, the comparison made here offers a good grasp of the physical size of the software under analysis. These metrics were collected using the *cloc*[6] utility. Blank lines and comments are ignored, so only the actual source code is accounted for. Additionally, we have taken into consideration only the source code modules implementing the functionalities that were compared in this work, in section 6.2.6.2 below. Source code modules implementing other functionalities were left out.

[6] https://github.com/AlDanial/cloc

Table 5 – Source code metrics for AVC and JBVRC.

Application		SLOC
AVC (C)		**4214**
JBVRC (C++)	Automatically generated	1597
	Components Library	2526
	Total	**4123**

Table 5 shows that the volume of source code automatically generated for JBVRC is considerably less than the legacy AVC code. Besides the differences between the C and C++ programming languages, the other plausible cause of such variations in size is that the JBVRC's code makes use of the pre-programmed actors from the D-SPADES components library. In other words, many of the operations that had to be programmed into the AVC's source code are previously available to JBVRC in the components library, so its code doesn't have to include these operations "in line". Furthermore, if we take into account the SLOC of the components library along with the JBVRC's automatically generated code, the figures get balanced, and it can be said that both applications have approximately the same physical size. It is worth noting that the code from the D-SPADES components library can be reused in other projects, therefore, for applications developed in the future, virtually no additional code needs to be handwritten, except possibly for new components that are added to the library.

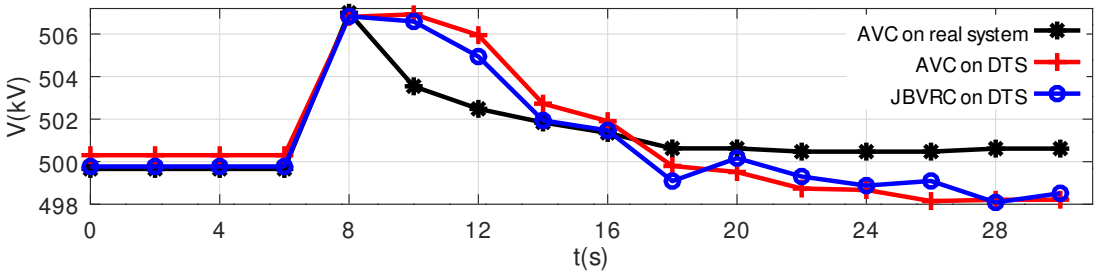
6.2.6.2 Functional Performance

We have evaluated a real system disturbance scenario that produces significant voltage variations at the power plant's 500 kV bus bar, by switching a 350 Mvar shunt capacitor bank at the nearby Foz do Iguaçu substation. Under the considered load condition and a number of synchronized generators, such switching caused an approximately 6 kV variation: voltage increases when the bank is turned on, and decreases when turned off. Figure 57 shows the data obtained from the plant's historical data, corresponding to the real-world event of the capacitor bank being switched on (curve in black). In this case, the AVC application was active and quickly brought the voltage back to the scheduled value of 500 kV.

We have replicated the same capacitor switching scenario on the DTS and verified that AVC response is reasonably consistent with the results observed in the real system, considering the limitations of the dynamic model implemented by the DTS, as shown in Figure 57 (curve in red). Finally, we have replaced AVC with the new JBVRC software executable on the DTS, and performed the same capacitor switching. The observed response is shown in Figure 57 (curve in

blue). These results are consistent with the AVC response under the same conditions, and also consistent with the continuous-time simulation performed by the *Ptolemy II* shown in Figure 50.

Figure 57 – 350 Mvar step: observed and simulated voltage performance.



Source: Neis *et al.* (2023).

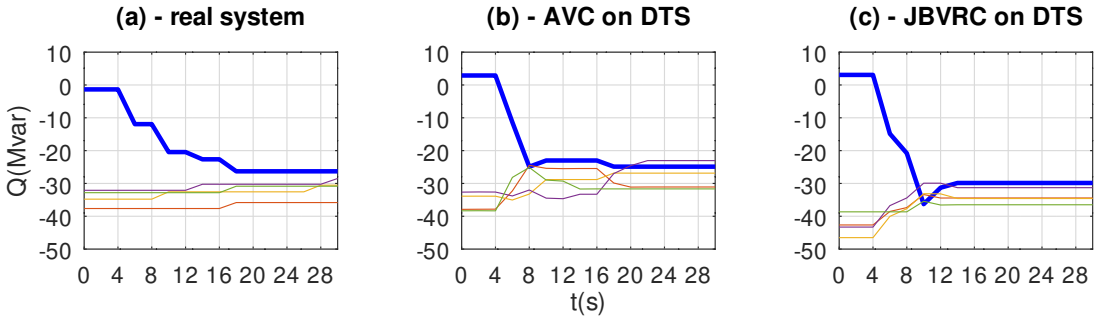
Other functional requirements, e.g., reactive power sharing among generators according to participation factors and the configured dead bands, have also been verified. Figure 58 illustrates the reactive power sharing functionality in action, for a real event consisting of the following scenario: five generating units were synchronized; four of them were initially operating under automatic control; the fifth unit was operating under manual voltage control, with a slightly different voltage setup, thus, it did not initially participate in the reactive power sharing. At t=6s, the fifth unit is put under automatic control and thus starts sharing reactive power. The reactive power sharing stops once the deviation from average reaches a configurable dead band, modeled by the "MvarDB" block from Figure 53.

Figure 58 shows the system behavior for this scenario. The data obtained from the plant's historian archive for the real event using the AVC application is depicted in (a): the unit under manual control, traced in blue, starts with its reactive power considerably above the remaining units and, after switched to automatic control, quickly drifts towards the others, which in turn are slightly increased so the total reactive power remains unaltered.

The results of a simulation of similar conditions, performed on the DTS, again using the AVC application are depicted in (b). In comparison, Figure 58 (c) shows the performance of the JBVRC application under similar conditions, also on the DTS simulator. In all three cases above it can be verified that the reactive power distribution has occurred consistently, respecting the configured dead band. This demonstrates that the JBVRC application, which was developed using the MDE-based approach and the code generation proposed in D-SPADES, achieved a similar performance when compared with the hand-crafted AVC application.

The results of these simulations suggest that D-SPADES provides an accurate mapping of high-level models into application source code, therefore being a feasible approach in practice.

Figure 58 – Reactive sharing: observed and simulated performance.



Source: Neis *et al.* (2023).

They also demonstrate that the software generated automatically from the AO model provides similar performance in comparison with its counterpart, i.e., the AVC software, which has been manually developed, tested, debugged, and improved over many years of operation. Additionally, besides the demonstrated feasibility of using D-SPADES in a real-world application, one can expect other project-related gains due to applying model-driven design techniques, e.g. reducing the effort and errors in the software coding phase, improving reuse, or shortening design time, as already widely suggested (FELIX *et al.*, 2020; VOELTER *et al.*, 2019; WEHRMEISTER *et al.*, 2014; YANG *et al.*, 2020; SANTOS *et al.*, 2020).

6.2.6.3 Computational Performance

We have conducted a comparison of computational performance metrics for both AVC and JBVRC under similar operational conditions. All source code files were compiled using the same compiler bundle, except that AVC uses the C language compiler, whilst JBVRC needs the C++ counterpart. In both cases, similar compiler optimization flags were employed. Both programs were executed and profiled during a five minutes interval, while performing control actions corresponding to the correction of the voltage disturbance described in Section 6.2.6.2. This particular comparison was performed exclusively using the tight integration strategy described in Section 5.4.3.1 (page 115). Metrics were collected using the standard host's operating system profiling tool, called *caliper* (HUNDT, 2000), and are shown in Table 6.

Table 6 – Profiling information for JBVRC and AVC.

Processor time (s)			**Memory pages**		
	AVC	JBVRC		AVC	JBVRC
user	0.112	0.162	shared	16641	25362
system	0.587	0.231	private	311	1072
total	0.699	0.393	weighted	820	1864

The processor usage metrics, measured in seconds of CPU time, suggests that the JBVRC executable spends fewer resources than AVC, particularly on system calls. The total CPU time (user+system) spent by JBVRC corresponds to approximately 56% of the time spent by AVC. In other words, it means that JBVRC performs the same task using roughly half the processing power required by AVC.

In terms of memory usage, however, JBVRC requires considerably more resources than AVC. The memory usage shown in Table 6, measured in pages of 4096 bytes, suggests that JBVRC requires more than double (227%) the storage resources compared to AVC, when the *weighted memory pages* metric is taken into account. This metric corresponds to the number of private pages for the process plus a proportion of the shared pages, weighted by the number of processes actually sharing each page. The number of weighted pages is considered a good estimate of the *load* imposed by the process onto system memory. In other words, JBVRC requires roughly twice the memory space to perform the same task as AVC. We consider these results perfectly acceptable, since although JBVRC uses more memory in comparison to AVC, the total amount used ($\sim$7 MiB) is still low for today's software applications. Additionally, we have to consider that D-SPADES adds a layer of abstraction between modeling and the source code. This additional layer adds some burden in terms of resource usage.

These results suggest that computationally efficient applications can be produced through the D-SPADES approach, despite the additional layer of abstraction and the corresponding extra libraries required at run time, when compared to the legacy applications developed using a general-purpose language.

6.3 CASE STUDY 2: ERG60 APPLICATION

We have applied D-SPADES to the development of a real-world application currently operating at the Itaipu Power Plant, designated as *Esquema de Redução de Geração em 60 Hz (ERG60)*[7]. The ERG60 application is part of a larger system, responsible for performing supplementary protective actions at the power plant's 60 Hz sector, and associated transmission network interconnecting it to the Brazilian grid. The ERG60 can be classified as a *discontinuous control*, i.e.: it is meant to bring the system back to normal after off-nominal conditions are detected. It operates in the timescale of several seconds to minutes (see Figure 19 from page 59). The D-SPADES software process described in Section 5.1 was fully exercised in the production of

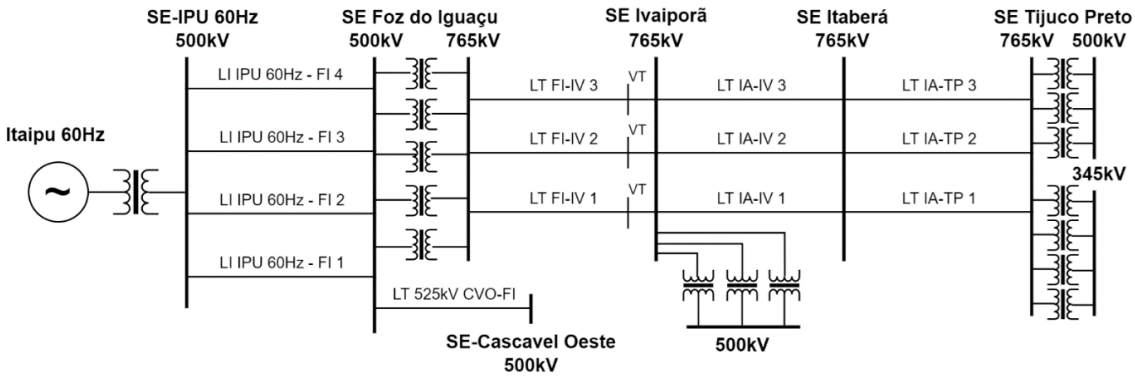
[7] ERG60 is the Portuguese acronym for 60 Hz sector's Generation Reduction Scheme.

the ERG60 application, including the successful deployment of the application in the production system. The application runs flawlessly since its deployment, in October/2022 (NEIS *et al.*, 2022).

6.3.1 Equipment and Systems Involved

Itaipu's 60 Hz sector has 7,000 MW installed capacity and is integrated into the Brazilian grid trough 500 kV and 765 kV transmission network, encompassing several substations containing numerous pieces of equipment such as transformers, shunt reactors, var compensators, lines, and series capacitors. Figure 59 illustrates a simplified view of this network, which geographically spans more than 900 km.

Figure 59 – Itaipu's 60 Hz sector and associated transmission network.



Source: (NEIS *et al.*, 2022).

This vast network, besides the standard equipment-based protective devices, is also equipped with a supplementary protection system, called *Sistema Especial de Proteção 765 kV (SEP-765)*[8]. The objectives of this system are to prevent contingencies in the network from bringing the whole grid to an unstable condition or risking the integrity of power equipment, including Itaipu's generating units. It does so by means of automatic control actions, such as: disconnecting (shedding) generating units at the Itaipu Power Plant, disconnecting transmission lines or busbars, and also performing controlled reduction of power production in order to eliminate equipment overload.

ERG60 is the specific action that reduces the power production at Itaipu Power Plant in order to eliminate such overload. ERG60 is implemented by a periodic subroutine associated

[8] SEP is the Portuguese acronym for Special Protective Scheme. According to Brazilian grid code (ONS - Operador Nacional do Sistema Elétrico, 2022), a SEP is *a system which, based on the detection of abnormal operational conditions or multiple contingencies, performs automatic actions in order to preserve the integrity of the power system, its equipment or transmission lines.*

with the 60 Hz sector's AGC.

6.3.1.1 Earlier Versions of ERG60

The demand for the ERG60 application was first identified back in 2012 when the risk of overload in the 500/765 kV transformers at the Foz do Iguaçu substation constituted a limiting factor for the power production in the 60 Hz sector, which could not be fully explored up to the installed capacity. Such limitations could be circumvented by implementing a special emergency control scheme that automatically (without any human intervention) detects such overload and reduces power production at the power plant, until the overload is eliminated. Thus the original ERG60 was developed, deployed and evaluated (NEIS *et al.*, 2012; NEIS *et al.*, 2012).

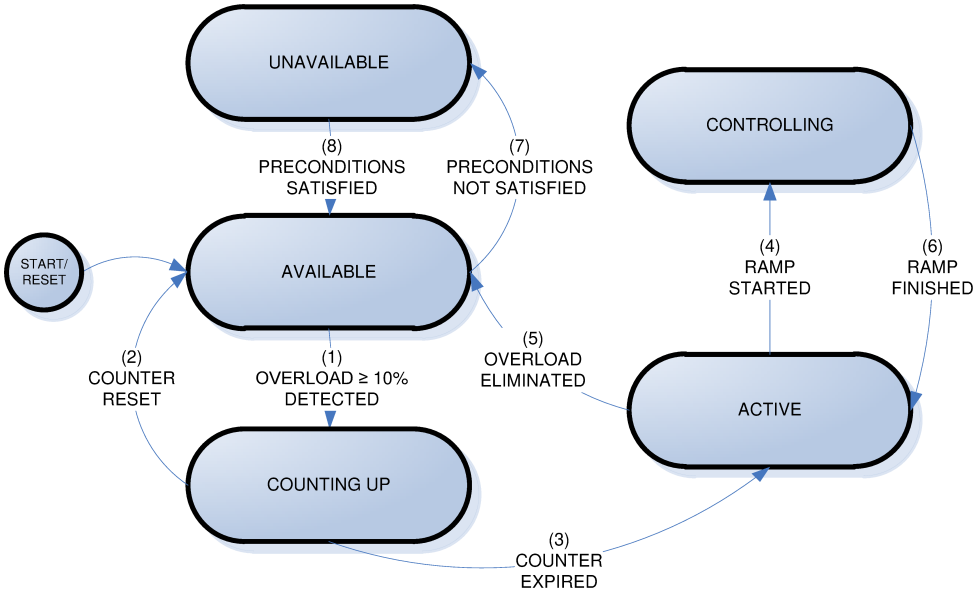
ERG60 worked by evaluating telemetered values received via ICCP from the National Operator. These values correspond to the electrical currents, measured at the low voltage transformer terminals, active and reactive power output, plus quality tags. The greater of the currents or apparent power output among the transformers is selected as reference, corresponding to the severest loading among them. The scheme was activated once an overload above 10% of the nominal current (or apparent power) was detected. The control actions consisted in issuing a command the 60 Hz AGC, so it would gradually reduce power plant generation through ramps of configurable rate and duration. By the end of the scheme's actuation, the overload on all the transformers shall be completely eliminated.

ERG60 was originally modeled using the FSM concept, as shown in Figure 60. This model, however, served only as an early design and documentation artifact: *model as sketch*, as previously discussed in Section 2.2 (page 30). The actual source code for the application was manually written based on this design and the documented requirements.

6.3.1.2 Evolution of ERG60

During the year 2022, a complete overhaul of the SEP-765 was conducted (PORTUGAL *et al.*, 2022; NEIS *et al.*, 2022). Along with this overhaul, the scope of the ERG60 was expanded. From this point on, ERG60's operation is triggered by the occurrence of an overload in any of the 765 kV transmission lines or any of the 500/765 kV transformers shown in Figure 59 (overloads in the range from 10% to 50%). Such occurrence is detected by one of the IEDs that compose the protective scheme and relayed to Itaipu's SCADA system by means of a

Figure 60 – FSM model for original ERG60 application.



Source: The author, based on (NEIS *et al.*, 2012).

pair of redundant RTUs. Therefore, the new ERG60 no longer monitors telemetered measured values from individual equipment. Instead, it receives a pair of redundant binary indications (status points), whose activated state indicates the presence of overload in at least one of the monitored lines or transformers. Such change required a major rewriting of the ERG60 source code, thus a good opportunity to apply D-SPADES. In order to differentiate the previous ERG60 implementation from the new one described here, we will, from now on, refer to it as the "Model-Driven ERG60 (MDERG)".

6.3.2 Overall Requirements of MDERG

The main requirements of MDERG are:

1. The scheme shall be automatically activated once an overload is indicated by the SEP-765 IEDs, and become inactive once the overload has been eliminated.

2. Once an overload indication is detected, the scheme shall wait for a configurable number of scan cycles for confirmation, before starting to issue commands. This shall be done in order to prevent the triggering of the scheme by spurious indications.

3. Once activated, the first action to be taken is to switch AGC mode to "REPA LOCAL" (splitter, local mode), if not already in this mode, and also cancel any ongoing ramp/rescheduling.

4. The module implementing the scheme shall command the 60 Hz AGC to gradually reduce power plant generation by means of ramps of configurable amplitudes and duration.

5. In case overload indication persists after a ramp is complete, the scheme shall remain active and command a new descending ramp.

6. The conditions determining the scheme (un)availability are: The scheme shall become unavailable whenever the AGC software is unavailable, and when the quality indication flags received from the IEDs (Telemetry Error, Deactivation, etc) are set.

7. An alarm shall be issued to indicate whenever the scheme starts issuing commands to reduce overload, and whenever the scheme becomes unavailable.

8. A toggle switch shall be implemented such that the scheme can be inhibited (turned off) by human intervention. This may be necessary for some specific power system conditions. This button shall also have the effect of canceling an ongoing activation and stopping the associated AGC ramp.

6.3.3 Proposed Architecture

MDERG, as well as ERG60, is tightly coupled with the SCADA. It is implemented as a periodically called subroutine, hooked to the main 60 Hz AGC execution cycle. Since the current Itaipu's AGC, as well as the ERG60, are implemented using the Fortran language, MDERG is also implemented in that language. For that, a different M2T template was developed, focused on the MDERG implementation, as well as a "lightweight" version of the D-SPADES components library, implemented in Fortran[9].

Complementarily to the conventions mentioned in Section 5.3.2.2 (page 109), the following criteria were established in order to facilitate integration with the AGC code, as well as isolating MDERG implementation from SCADA code:

- As stated before, MDERG is a subroutine periodically called by Itaipu's AGC. The behavior of this subroutine is governed by an FSM, similar to the ERG60's shown in Figure 60. This subroutine is called "ECEIPU".

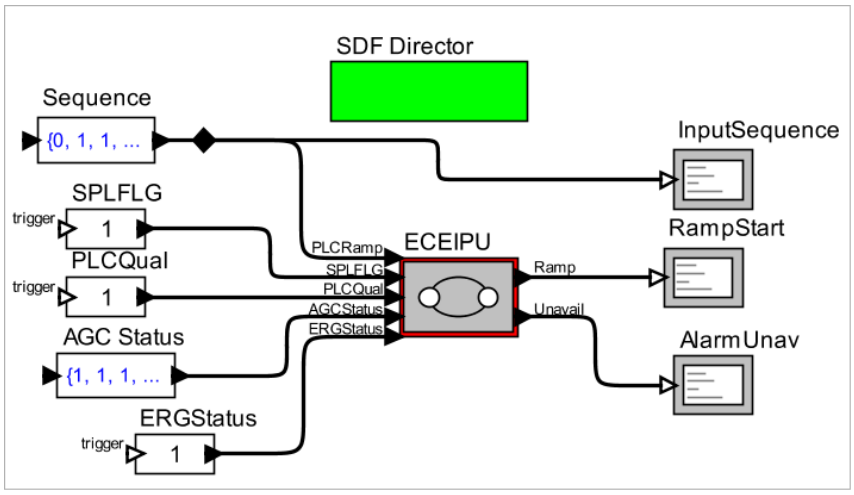
[9] Although we acknowledge that the C++ version of D-SPADES components library could be used, that would add extra overhead on the AGC executable, both in memory usage as well as executable size, due to the linkage with the C++ library.

- ECEIPU is called at the end of the main AGC execution cycle, and the input data needed by the FSM, represented in the model as input ports, is passed to the sub-routine as arguments.
- Output data, represented in the model as output ports, are implemented as subroutine calls. Thus, for every output port in the model, a callback subroutine has to be implemented in the D-SPADES Fortran library.

Figure 61 shows the ECEIPU subroutine, modeled in the *Ptolemy II* environment, with its inputs and outputs.

- **PLCRamp** (IN): Status indication of overload condition (Yes/No) from the SEP-765 IED.
- **PLCQual** (IN): Telemetry quality indication from the redundant RTU communication.
- **AGCStatus** (IN): The current state of the AGC controller (ON/OFF).
- **SPLFLG** (IN): Binary indication of the current status of ramp execution from the AGC (ON/OFF).
- **ERGStatus** (IN): Switch controlling the state of the application (ON/OFF).
- **Ramp** (OUT): Starts (or stops) the execution of a ramp, and also issues the corresponding alarm.
- **AlarmUnav** (OUT): Issues the alarm indicating that the MDERG is unavailable, and its return to normality.

Figure 61 – ECEIPU subroutine model: inputs and outputs.

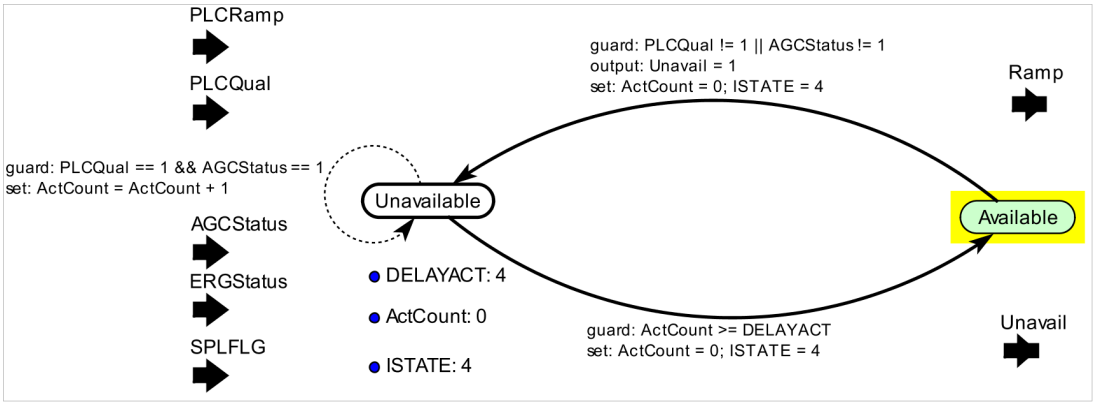


Source: The author.

6.3.4 Cyber-Physical Process and Application Model

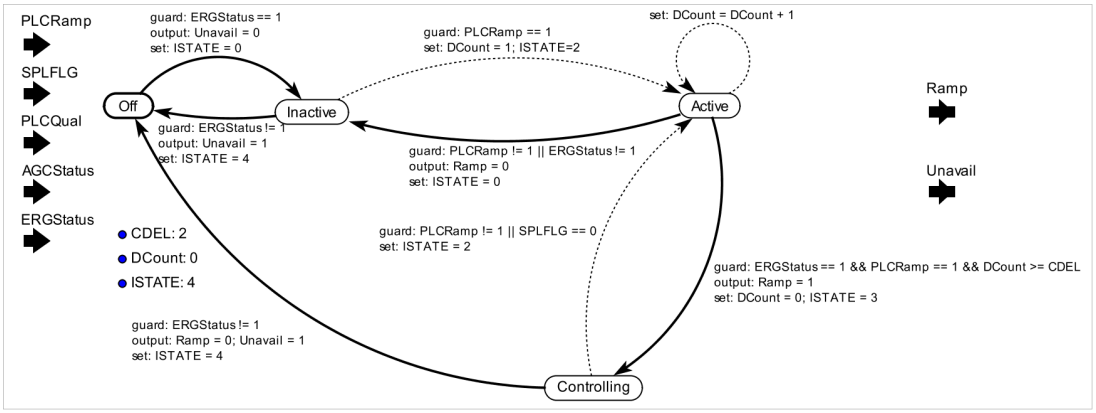
MDERG runtime environment consists of a periodical scheduler, responsible for executing it every 4s, and the infrastructure that provides input/output data and executes actions. In the *Ptolemy II* environment, this can be represented as an SDF model providing sequences of data as inputs, and displaying the function's output in order to perform early model validation, as shown in Figure 61. The "ECEIPU" block shown in this figure consists of the hierarchical FSM model represented in Figures 62 and 63.

Figure 62 – FSM hierarchical model of the ECEIPU subroutine for MDERG.



Source: The author.

Figure 63 – Model refinement for "Available" state.



Source: The author.

The hierarchical FSM, whose higher-level model is depicted in Figure 62, starts its execution in the "Unavailable" state. From this state, its guard expression determines that it shall transition to the "Available" state only if "$PLCQual == 1$" and "$AGCStatus == 1$", which means: the telemetry quality associated with the redundant RTU must be "good" (at least one RTU is up) and the 60 Hz AGC must be turned on. In case this transition is activated, an

output expression is also executed, in this case: “AlarmUnav=0”. According to the conventions previously established in Section 6.3.3, a corresponding subroutine is called passing “0” as the argument, meaning that the alarm condition for MDERG availability is reset (return to normal alarm).

Once in the “Available” state, the FSM might transition back to “Unavailable” if the specified guard expression is satisfied, reciprocally to the transition described above. This FSM also has a model refinement defined for the “Available” state, as shown in Figure 63, meaning that, in this state, another instance of an FSM is executed. The operation of this instance is governed by the model represented in Figure 63, which essentially represents the behavior that satisfies the requirements listed in Section 6.3.2.

6.3.4.1 Model Transformation

The model transformation for the MDERG application is performed using a specific template targeted to produce Fortran code. An excerpt from this template is shown in Listing 6.1.

Listing 6.1 – Excerpt of Acceleo MTL template for Fortran language.

```
1 [comment encoding = UTF-8 /]
2 [module generateFortranFSM('http://www.example.org/EMSML')]
3 [comment This is the MAIN TEMPLATE. /]
4 [template public generateElement(anEntity : entity) ? (self.name = 'ECEIPU') ]
5 [comment @main/]
6 [file ('OUTSTREAM.txt', false, 'UTF-8')]
7 [anEntity.genFSMStructure()/]
8 [/file]
9 [/template]
10 [**
11  * This template generates all the FSM structure and subroutines. It can be called
       recursively
12 */]
13 [template private genFSMStructure(anEntity : entity)
14  {className : String = anEntity.name.replaceAll(' ', '_').trim();} ]
15 [if anEntity.class = 'ptolemy.domains.modal.modal.ModalModel' or anEntity.class = 'ptolemy
      .domains.modal.modal.ModalRefinement']
16 Entity: [className /] IS a ModalModel!
17 [for (anEntityIter : entity | anEntity.entity)]
18 [if anEntityIter.class = 'ptolemy.domains.modal.modal.ModalController']
19 Found ModalController: [anEntityIter.name.replaceAll(' ', '_').trim() /] INSIDE: [
      className /]
20 [anEntityIter.genFSMController()/]
21 [/if]
22 [if anEntityIter.class = 'ptolemy.domains.modal.modal.ModalRefinement']
```

```
23 Found ModalRefinement: [anEntityIter.name.replaceAll(' ', '_').trim() /] INSIDE: [
       className /]
24 [comment  Recursively call the genFSMStructure() template /]
25 [anEntityIter.genFSMStructure() /]
26 [/if]
27 [/for]
28 [else]
29 ERROR! The model you tried to process is not an FSM model.
30 [/if]
31 [/template]
32 [comment This template substitytes C-style binary operators with F77 operators. /]
33 [template private subsF77 (aString : String) post( trim() )]
34 [aString.substituteAll('>=', '.GE.').substituteAll('<=', '.LE.').substituteAll('==', '.EQ.
       ').substituteAll('!=', '.NE.').substituteAll('||', '.OR.').substituteAll('&&', '.AND.'
       ).substituteAll('>', '.GT.').substituteAll('<', '.LT.') /]
35 [/template]
```

6.3.5 Deployment and Test Results

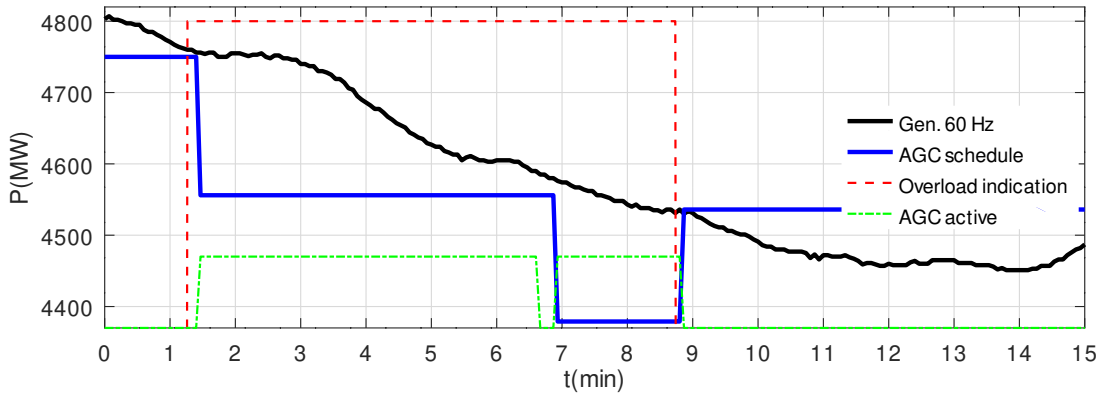
The new AGC executable containing the MDERG was initially validated using the DTS, with satisfactory results. The next step consisted in deploying the executable in the production SCADA/EMS and executing integration tests with the whole SEP-765 in real operational conditions. During these tests, no software defects were detected. It is worth noticing that the code produced is perfectly readable and comprehensible, thus it can be easily maintained and debugged, if necessary.

6.3.5.1 Functional Performance

Below we describe one of the tests executed, which consists in simulating an overload event, by forcing the scheme's IEDs to indicate this condition, although the equipment was operating under normal loading. For this test, the MDERG was configured to perform reduction ramps of 200 MW, with a 5 min duration.

Figure 64 shows the behavior of the 60 Hz sector's power production during a 15 min interval while the test took place. The plant's 60 Hz sector was producing approximately 4750 MW (trace in black) when the overload indication was raised (red dashed line). The trace in blue shows that the AGC target schedule was changed to approximately 4550 MW, and the ramp command was activated (SPLFLG variable, green dashed line). The first 200 MW ramp was completed after 5 min, with the power production reaching approximately 4580 MW. Since the

Figure 64 – Integration test performing generation reduction.



Source: Neis *et al.* (2023).

overload indication was still active, MDERG set the scheduled generation to 4380 MW, starting a new ramp. This behavior, according to requirement (5), is modeled in the FSM shown in Figure 63 by the guard expressions conditioning the transitions back and forth from state “Active” to “Controlling”. Approximately 2 min after the second ramp started, the overload indication was lowered, so MDERG changed the scheduled generation to 4536 MW (the current measured power) and canceled the ramp. The behavior described above is considered correct according to the established requirements of the function, which was therefore deployed in the production system.

6.4 CHAPTER SUMMARY

In this chapter, we have demonstrated how D-SPADES can be applied to the development of executable software. Initially, a simple toy application was modeled, transformed into C++ source code, and executed as a console application, in order to demonstrate the D-SPADES workflow. Following, we have demonstrated the viability of D-SPADES by modeling the JBVRC after the AVC, a real-world application from the Itaipu Power Plant. The JBVRC was integrated into Itaipu’s DTS and tested in comparison to the legacy AVC, showing satisfactory results. The JBVRC has served as a proof-of-concept for D-SPADES, demonstrating that it can produce software that executes correctly and efficiently. Finally, we have applied D-SPADES to the development of the MDERG application, which was deployed to the production SCADA/EMS at the Itaipu Power Plant.

7 CONCLUSIONS AND FUTURE WORK

In this work, we have conducted a thorough investigation regarding techniques and tools that can be applied in the development of EMS applications. Based on the initially stated research questions, the thesis was developed and reached the following achievements:

7.1 ANSWERS TO RESEARCH QUESTIONS

Regarding the questions raised in Section 1.1.1, based on the argumentation and the results reported in the previous chapters, the following answers are formulated:

From the Energy Management perspective:

1. **Can we choose a satisfactory format for modeling EMS applications?**
   We propose to use actor-oriented models since they can easily accommodate the block diagram and transfer function paradigms largely used in traditional control theory, as well as other useful abstractions such as FSMs. Several modeling and simulation tools applied in the power systems domain are block diagram based design environments, suggesting that the paradigm has widespread acceptance in the area. We have collected evidence suggesting that this paradigm is widely used for modeling power equipment and system behavior, as well as to model the control applications that interact with them. In the particular case of Itaipu, we have access to a significant volume of documentation and simulation models based on block diagrams and transfer functions. We have demonstrated by means of a proof-of-concept application, that these legacy models can be easily converted into EMSML and then further processed through D-SPADES work-flow in order to produce executable software. The existence of such legacy models thus facilitates the adoption of the D-SPADES approach. Other companies in the power sector are likely to have their documentation and simulation models also expressed in similar formats, and so they too can benefit from an approach such as D-SPADES.

2. **How can the proposed approach contribute to improving model verification and validation?**
   The adoption of actor-oriented models compatible with the *Ptolemy II* package offers the possibility of modeling and simulating both the physical process and the cyber counterpart

– the controller. Therefore the controller model can be incrementally built and validated in the actor-oriented simulation environment. Using models that were previously validated as input to the software generation step, according to MDE literature, shall significantly improve the chances of producing an application that is correct "by design".

3. **Is it possible to automate the transformation of these models into executable artifacts?**

   We have demonstrated that abstract actor-oriented models can be processed and transformed into source code through M2T transformations. Additionally, source code for different platforms or programming languages can be produced by creating new transformations (or adapting existing ones). Thus we can use the same abstract model to produce source code for different platforms. Since abstracting always involves simplifications, and therefore omitting some implementation details, a gap exists between the model specification and the target implementation that needs to be filled in order to produce a functional executable application. This gap is filled by the D-SPADES components library, which implements the building blocks necessary for realizing EMS applications.

4. **Is it possible to integrate these executable artifacts with both the existing and future generations of the centralized control systems?**

   We have demonstrated that the software produced from the high-level application models can be integrated into a real-world SCADA/EMS. We have presented, in Chapter 6, both a proof-of-concept application, as well as a production-level application integrated with the current Itaipu's SCADA/EMS. Integration with future generations of SCADA/EMS packages may be facilitated by the adoption of standardized middleware, but so far we have integrated the aforementioned applications to only one SCADA/EMS product line.

From the Software Engineering perspective:

1. **Is it possible to model the behavior and structure of EMS applications using a high-level domain-specific language? What kind of language is indicated?**

   The answer to this question is basically the same as the one provided above for the Energy Management perspective: actor-oriented models are an appropriate choice for a high-level representation of EMS applications' intended behavior. We have also shown that these models can be processed and automatically transformed into source code through M2T

transformations. From a Software Engineering perspective, this is an interesting conclusion, since most of the modeling languages currently seen in MDE approaches are UML-based.

2. **Is it possible to reuse existing/legacy artifacts – specifications, block diagrams or programs written in third generation languages?**
   We have shown that high-level specifications like block diagrams and transfer functions can be easily reused or adapted to the EMSML language, for instance: in the modeling of the JBVRC shown in Figure 48 (page 129). Legacy source code could be reused in the components library, but practical cases of such reuse are still to be determined.

3. **Which transformation techniques and tools can be used?**
   We have used model-to-text transformations, described in a template-based language. The Acceleo tool and the MTL transformation language were used for that effect. In addition, the EMF Ecore is the format adopted for metamodeling. Model-to-Model transformations, although not used in the development of this work, may be used as a "bridge" between different modeling tools used in the design process, e.g.: transforming a Simulink/Stateflow model into EMSML.

4. **How can these software artifacts be integrated with existing commercial supervisory control systems?**
   This integration can be achieved using either the tight or loose integration architectures we proposed in Section 5.4.3.1 (page 115). We have demonstrated the feasibility of both approaches through a proof-of-concept application described in Chapter 6, but currently adopt a tightly integrated approach for production applications.

7.2 THESIS CONTRIBUTIONS

In this thesis, we have proposed D-SPADES: an MDE-based approach to the development of EMS applications, focused on hydropower plants. This approach consists of processes, languages, and tools that are tailored to develop and maintain such applications. D-SPADES is being developed to address the immediate necessities we have observed in our professional activities within the Itaipu Power Plant. We believe, however, that the approach is reproducible and widely applicable in the power industry and other related areas, since the paradigms like block diagrams and transfer functions, widely used for modeling power plant equipment and associated systems, are also useful for modeling a vast number of industrial processes.

It is important to point out that, by proposing D-SPADES, we are not advocating that traditional development approaches, such as those using 3GLs, are to be abandoned. In other words, D-SPADES is not a replacement for traditional approaches, but rather a complement. By offering a higher level of abstraction to the computing infrastructure, we expect to allow better integration of domain experts into the software development process, without requiring them to have deep knowledge of the low-level computer programming models. This integration is promoted through a better separation of roles:

Domain Specialists concentrate on the process models using expressive, domain-specific, modeling tools; but not necessarily having deep knowledge of the computer's lower-level programming model, language, and tools. As discussed in Section 2.1 (page 26), they only need knowledge of the layers of abstraction immediately above (the power system) and below (D-SPADES).

Software Developers can work more intensively developing model transformations and components for the D-SPADES library, providing support for the domain specialist; they are required to have deep knowledge of the MDE tools and programming models, but little knowledge of the power system domain.

Therefore D-SPADES has shown several advantages in comparison with a traditional development approach for SCADA/EMS applications, among them we can emphasize: (i) it provides a highly abstract, domain-specific modeling language (EMSML), suitable for the power systems specialists to participate actively in the development process; (ii) early model validation is inherently part of the design process since the models can be tested and evolved before the source code is produced. (iii) generated code is free of programming errors that might appear when the software is manually programmed based on the human interpretation of specifications.

In such an approach, domain specialists and software developers play more integrated, yet complementary roles: power engineers can focus on high-level problem solving, using DSML and simulation tools, while software developers concentrate on the support process, like providing an adequate set of actors and transformations for automated code production. Comparatively, in traditional approaches, any redesign or even minor changes in the application model inevitably requires refactoring, testing, and debugging software modules written in the chosen target programming language. With the D-SPADES approach, on the other hand, transformations and component libraries previously tested and validated are re-used. Newly added components can be unit-tested, and new transformations can also be previously validated using test models. Thus a complete application design can be conducted by power systems specialists, producing an

executable program ready to be integrated with the underlying SCADA system without manually writing a single line of code in conventional programming languages.

Our results reinforce the idea that early model validation associated with automatic code generation is an effective approach to high-quality software development. We also expect that D-SPADES will increase productivity and facilitate software evolution, for instance, when adding new requirements to a given application or upgrading the base SCADA platform. Additionally, D-SPADES explicitly addresses the problem of integrating EMS applications with commercial SCADA products. Such integration can be achieved either through a tightly-coupled architecture, using proprietary SCADA APIs, or a loosely-coupled approach using standard middleware such as OPC-UA.

7.2.1 Publications

During the development of the research associated with this thesis, we have published the following related works:

1. NEIS, P.; WEHRMEISTER, M.A.; MENDES, M.F. “Model driven software engineering of power systems applications: Literature review and trends”. **IEEE Access**, v. 7, p. 177761–177773, 2019. ISSN 2169-3536. http://dx.doi.org/10.1109/ACCESS.2019.2958275.

2. NEIS, P.; TOCHETTO, A. P.; RAMÍREZ, R. J. G.; COSTA, C. H. C.; WEHRMEISTER, M. A. “Integração do esquema de redução de geração em Itaipu 60 Hz com o novo SEP do sistema de transmissão em 765 kV: uma abordagem de engenharia guiada por modelos”. In: **XVII EDAO - Encontro para Debates de Assuntos de Operação**. São Paulo - SP: [s.n.], 2022. p. 1–9.

3. NEIS, P.; WEHRMEISTER, M. A.; MENDES, M. F.; PESENTE, J. R. Applying a model-driven approach to the development of power plant SCADA/EMS software. **International Journal of Electrical Power & Energy Systems**, V. 153, p. 109336, 2023. ISSN 0142-0615. https://doi.org/10.1016/j.ijepes.2023.109336.

7.3 FUTURE WORK

The next logical step in the development of this work is to consolidate and promote the adoption of D-SPADES within Itaipu and possibly other partners. Among the imminent applications of the approach, we can mention the development and integration of applications for the new SCADA/EMS, shipped with the Itaipu's modernization contract[1] and the development of improved simulation modules for the DTS. Along with the development of new projects, we will be able to promote the involvement of power systems specialists with no background in software development, thus allowing for a more precise assessment of the true potentials of D-SPADES. In complement, we intend to add support for a larger number of actors and MoCs into the D-SPADES components library, as well as a correspondent set of automated unit tests, so it can support larger projects. Additionally, the applicability of the D-SPADES approach in applications other than SCADA/EMS can be evaluated, for instance: in the development of applications running on IED and/or PLC platforms, performing special control or protective actions, like the ECCANDE project (GODOY *et al.*, 2023).

As future research, a thorough evaluation of the approach based on well-known standards must be performed, such as the ISO 9241-11 quality model, regarding aspects such as scalability, usability, and efficiency.

[1] http://www.itaipu.gov.br/en/technology/technological-upgrade

REFERENCES

ABB Inc. **Hydro power – Intelligent solutions for hydroelectric power plant controls**. Burlington, ON, Canada: *[s.n.]*, 2016. ABB information brochure.

AGARWAL, P. K.; DE, D.; RATHOUR, H. K. Modernization of multi location live SCADA system - a case study. *In*: **2016 National Power Systems Conference (NPSC)**. *[S.l.: s.n.]*, 2016. p. 1–6.

AGHA, Gul. Concurrent object-oriented programming. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 33, n. 9, p. 125–141, Sep. 1990. ISSN 0001-0782. Available at: https://doi.org/10.1145/83880.84528.

ANDRÉN, F.; BRÜNDLINGER, R.; STRASSER, T. IEC 61850/61499 control of distributed energy resources: Concept, guidelines, and implementation. **IEEE Transactions on Energy Conversion**, v. 29, n. 4, p. 1008–1017, Dec 2014. ISSN 0885-8969.

ANDRÉN, Filip; STIFTER, Matthias; STRASSER, Thomas. Towards a semantic driven framework for smart grid applications: Model-driven development using CIM, IEC 61850 and IEC 61499. **Informatik-Spektrum**, v. 36, n. 1, p. 58–68, Feb 2013. ISSN 1432-122X. Available at: https://doi.org/10.1007/s00287-012-0663-y.

ANDRÉN, F.; STRASSER, T.; KASTNER, W. Model-driven engineering applied to Smart Grid automation using IEC 61850 and IEC 61499. *In*: **2014 Power Systems Computation Conference**. *[S.l.: s.n.]*, 2014. p. 1–7.

ANDRÉN, F.; STRASSER, T.; KASTNER, W. From textual programming to IEC 61499 artifacts: Towards a model-driven engineering approach for smart grid applications. *In*: **2015 IEEE 13th International Conference on Industrial Informatics (INDIN)**. *[S.l.: s.n.]*, 2015. p. 1524–1530. ISSN 1935-4576.

ANDRÉN, F.; STRASSER, T.; ROHJANS, S.; USLAR, M. Analyzing the need for a common modeling language for smart grid applications. *In*: **2013 11th IEEE International Conference on Industrial Informatics (INDIN)**. *[S.l.: s.n.]*, 2013. p. 440–446. ISSN 1935-4576.

ANDRÉN, F. P.; STRASSER, T.; KASTNER, W. Applying the SGAM methodology for rapid prototyping of smart grid applications. *In*: **IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society**. *[S.l.: s.n.]*, 2016. p. 3812–3818.

ANDRÉN, Filip Pröstl; STRASSER, Thomas I.; KASTNER, Wolfgang. Engineering smart grids: Applying model-driven development from use case design to deployment. **Energies**, v. 10, n. 3, 2017. ISSN 1996-1073. Available at: https://www.mdpi.com/1996-1073/10/3/374.

ANEEL - Agência Nacional de Energia Elétrica. **Sistema de Informações de Geração - SIGA**. 2022. Accessed: 2022-December-23. Available at: https://www.gov.br/aneel/pt-br/centrais-de-conteudos/relatorios-e-indicadores/geracao.

ARYA, Yogendra; KUMAR, Narendra. Design and analysis of BFOA-optimized fuzzy PI/PID controller for AGC of multi-area traditional/restructured electrical power systems. **Soft Comput.**, Springer-Verlag, Berlin, Heidelberg, v. 21, n. 21, p. 6435–6452, Nov. 2017. ISSN 1432-7643. Available at: https://doi.org/10.1007/s00500-016-2202-2.

AZEVEDO, Gilberto Pires; OLIVEIRA-FILHO, Ayru. Control centers with open architectures [power system ems]. **Computer Applications in Power, IEEE**, v. 14, p. 27 – 32, 11 2001.

AZIZI, S. M.; KHAJEHODDIN, S. A. Designing decentralized load-frequency controllers: An optimization approach for synchronous generators in islanded grids. **IEEE Industry Applications Magazine**, v. 24, n. 2, p. 67–74, March 2018. ISSN 1077-2618.

BAEK, S. M. Design of robust voltage control system for improving transient and voltage stability on distributed generation expansion. *In*: **2014 IEEE Industry Application Society Annual Meeting**. *[S.l.: s.n.]*, 2014. p. 1–8. ISSN 0197-2618.

BAEK, S. M.; NAM, S.; SONG, J.; LEE, J.; KIM, T.; SHIN, J. Design of advanced voltage management system including manual operation mode via real-time digital simulator. **IEEE Transactions on Industry Applications**, v. 49, n. 4, p. 1817–1826, July 2013. ISSN 0093-9994.

BEUS, Mateo; PANDžIć, Hrvoje. Practical implementation of a hydro power unit active power regulation based on an mpc algorithm. **IEEE Transactions on Energy Conversion**, v. 37, n. 1, p. 243–253, 2022.

BEVRANI, H. **Robust Power System Frequency Control**. *[S.l.]*: Springer US, 2008. (Power Electronics and Power Systems). ISBN 9780387848785.

BEVRANI, H.; DANESHFAR, F.; HIYAMA, T. A new intelligent agent-based AGC design with real-time application. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, v. 42, n. 6, p. 994–1002, Nov 2012. ISSN 1094-6977.

BEVRANI, H.; HIYAMA, T. On load-frequency regulation with time delays: Design and real-time implementation. **IEEE Transactions on Energy Conversion**, v. 24, n. 1, p. 292–300, March 2009. ISSN 0885-8969.

BJÖRKMAN, Gunnar; SOMMESTAD, Teodor; EKSTEDT, Mathias; HADELI, Hadeli; ZHU, Kun; CHENINE, Moustafa. **SCADA system architectures**. 2010. Developed within the VIKING Consortium. QC 20150213.

BOGODOROVA, T.; SABATE, M.; LEÓN, G.; VANFRETTI, L.; HALAT, M.; HEYBERGER, J. B.; PANCIATICI, P. A Modelica power system library for phasor time-domain simulation. *In*: **IEEE PES ISGT Europe 2013**. *[S.l.: s.n.]*, 2013. p. 1–5. ISSN 2165-4816.

BOX, G.E.P.; DRAPER, N.R. **Response Surfaces, Mixtures, and Ridge Analyses**. *[S.l.]*: Wiley, 2007. (Wiley Series in Probability and Statistics). ISBN 9780470053577.

BRAMBILLA, Marco; CABOT, Jordi; WIMMER, Manuel. **Model-Driven Software Engineering in Practice: Second Edition**. 2nd. ed. *[S.l.]*: Morgan & Claypool Publishers, 2017. ISBN 1627057080.

BROOKS, Christopher; BUCK, Joseph; CHEONG, Elaine; Davis-II, John S.; DERLER, Patricia; FENG, Thomas Huining; GALICIA, Geroncio; GOEL, Mudit; HA, Soonhoi; LEE, Edward A.; LIU, Jie; LIU, Xiaojun; MESSERSCHMITT, David; MULIADI, Lukito; NEUENDORFFER, Stephen; REEKIE, John; RODIERS, Bert; SMYTH, Neil; XIONG, Yuhong; ZHENG, Haiyang. Software architecture. *In*: PTOLEMAEUS, Claudius (Ed.). **System Design, Modeling, and Simulation using Ptolemy II**. Ptolemy.org, 2014. Available at: http://ptolemy.org/books/Systems.

BROOKS, Christopher; LEE, Edward A.; NEUENDORFFER, Stephen; REEKIE, John. Building graphical models. *In*: PTOLEMAEUS, Claudius (Ed.). **System Design, Modeling, and Simulation using Ptolemy II**. Ptolemy.org, 2014. Available at: http://ptolemy.org/books/Systems.

BROOKS JR, Frederick. No Silver Bullet – essence and accidents of software engineering. **IEEE Computer**, v. 20, p. 10–19, 04 1987.

BRUINENBERG, Jan; COLTON, Larry; DARMOIS, Emmanuel; DORN, John; DOYLE, John; ELLOUMI, Omar; ENGLERT, Heiko; FORBES, Raymond; HEILES, Jürgen; HERMANS, Peter; USLAR, Mathias. **CEN -CENELEC - ETSI: Smart Grid Coordination Group - Smart Grid Reference Architecture Report 2.0**. *[S.l.: s.n.]*, 2012.

BUTCHER, Paul. **Seven Concurrency Models in Seven Weeks: When Threads Unravel**. 1st. ed. *[S.l.]*: Pragmatic Bookshelf, 2014. ISBN 1937785653.

CALOVIC, Milan; JELIC, Nenad. Joint bus voltage/reactive generation control in multimachine power plants: a multivariable approach. **International Journal of Electrical Power & Energy Systems**, v. 14, n. 6, p. 393 – 401, 1992.

CARDOSO, Janette; LEE, Edward A.; LIU, Jie; ZHENG, Haiyang. Continuous-time models. *In*: PTOLEMAEUS, Claudius (Ed.). **System Design, Modeling, and Simulation using Ptolemy II**. Ptolemy.org, 2014. Available at: http://ptolemy.org/books/Systems.

CASTRO, F.; PESCINA, M.; LLORT, G. Reliability improvements of the Guri Hydroelectric Power Plant computer control system AGC and AVC. **IEEE Trans. Energy Convers.**, v. 7, n. 3, p. 447–452, 1992.

CHHOKRA, Ajay; BARRETO, Carlos; DUBEY, Abhishek; KARSAI, Gabor; KOUTSOUKOS, Xenofon. Power-attack: A comprehensive tool-chain for modeling and simulating attacks in power systems. *In*: **Proceedings of the 9th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems**. New York, NY, USA: Association for Computing Machinery, 2021. (MSCPES '21). ISBN 9781450386081. Available at: https://doi.org/10.1145/3470481.3472705.

COHEN, A.; KEMPER, F.; Dy Liacco, T.; CÁCERES, D. The SCADA/EMS system of the ITAIPU hydroelectric powerplant. *In*: **IFAC Proceedings Volumes**. Seoul, Korea: *[s.n.]*, 1989. v. 22, n. 9, p. 279–283. ISSN 1474-6670. IFAC Symposium on Power Systems and Power Plant Control 1989, Seoul, Korea, 22-25 August 1989.

CORERA, J. M.; MARTÍ, J.; OJINAGA, Z.; LEX, W.; KUHLMANN, A. Implementation of a new scada/ems/dms in a large utility, integrated with corporate information systems. *In*: **CIRED 2005 - 18th International Conference and Exhibition on Electricity Distribution**. *[S.l.: s.n.]*, 2005. p. 1–5.

CORSI, S.; POZZI, M.; SABELLI, C.; SERRANI, A. The coordinated automatic voltage control of the Italian transmission grid-part I: reasons of the choice and overview of the consolidated hierarchical system. **IEEE Transactions on Power Systems**, v. 19, n. 4, p. 1723–1732, Nov 2004. ISSN 0885-8950.

CORSI, S.; POZZI, M.; SFORNA, M.; DELL'OLIO, G. The coordinated automatic voltage control of the Italian transmission grid-part II: control apparatuses and field performance of the consolidated hierarchical system. **IEEE Transactions on Power Systems**, v. 19, n. 4, p. 1733–1741, Nov 2004. ISSN 0885-8950.

COTRIM, John Reginald. **Itaipu hydroeletric project: engineering features**. *[S.l.]*: Itaipu Binacional, 1994. ISBN 85-85263-02-4.

DÄNEKAS, Christian; NEUREITER, Christian; ROHJANS, Sebastian; USLAR, Mathias; ENGEL, Dominik. Towards a model-driven-architecture process for smart grid projects. *In*: BENGHOZI, Pierre-Jean; KROB, Daniel; LONJON, Antoine; PANETTO, Hervé (Ed.). **Digital Enterprise Design & Management**. Cham: Springer International Publishing, 2014. p. 47–58. ISBN 978-3-319-04313-5.

Electric Power Research Institute. Common Information Model Primer - Fourth Edition. **Electric Power Research Institute (EPRI)**, 2018.

EREMIA, M.; SHAHIDEHPOUR, M. **Handbook of Electrical Power System Dynamics: Modeling, Stability, and Control**. *[S.l.]*: Wiley, 2013. (IEEE Press Series on Power Engineering). ISBN 9781118516065.

FELIX, Eder; LOPES, Denivaldo; JR., Osvaldo Sousa. A framework based on model driven engineering and model weaving to support data-driven interoperability for smart grid applications. *In*: **Proceedings of the 2020 European Symposium on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2020. (ESSE 2020), p. 30–36. ISBN 9781450377621. Available at: https://doi.org/10.1145/3393822.3432341.

FENG, Thomas Huining; LEE, Edward A.; LIU, Xiaojun; TRIPAKIS, Stavros; ZHENG, Haiyang; ZHOU, Ye. Modal models. *In*: PTOLEMAEUS, Claudius (Ed.). **System Design, Modeling, and Simulation using Ptolemy II**. Ptolemy.org, 2014. Available at: http://ptolemy.org/books/Systems.

FRANCE, R.; RUMPE, B. Model-driven development of complex software: A research roadmap. *In*: **Future of Software Engineering (FOSE '07)**. *[S.l.: s.n.]*, 2007. p. 37–54.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software**. USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0201633612.

GODOY, Jose Maria Barua; de Oliveira, Robson Almir; AGUAYO, Gustavo; RODRIGUEZ, Elisandro; SZOSTAK, Alfredo Javier Mezger; SANTOS, Jhonatan Andrade dos; TOCHETTO, André Pagani; RIOS, Manuel Leonardo Sosa; GALASSI, Paulo Henrique; PESENTE, Jonas Roberto; RAMOS, Rodrigo Andrade. The eccande project: Design, field implementation, and operation of a special protection scheme based on synchronized phasor measurements. **IEEE Transactions on Power Delivery**, v. 38, n. 3, p. 1780–1787, 2023.

GÓMEZ, F. J.; VANFRETTI, L.; OLSEN, S. H. CIM-compliant power system dynamic model-to-model transformation and Modelica simulation. **IEEE Transactions on Industrial Informatics**, v. 14, n. 9, p. 3989–3996, Sep. 2018. ISSN 1551-3203.

GRIGSBY, L.L. **The Electric Power Engineering Handbook, Five Volume Set, Second Edition**. *[S.l.]*: Taylor & Francis, 2007. ISBN 9780849392931.

HAQ, Enamul; ROTHLEDER, Mark; MOUKADDEM, Bassem; CHOWDHURY, Sirajul; ABDUL-RAHMAN, Khaled; FRAME, James G.; MANSINGH, Ashmin; TEREDESAI, Tushar; WANG, Norman. Use of a grid operator training simulator in testing new real-time market of California ISO. *In*: **2009 IEEE Power Energy Society General Meeting**. *[S.l.: s.n.]*, 2009. p. 1–8.

HARRAND, Nicolas; FLEUREY, Franck; MORIN, Brice; HUSA, Knut. Thingml: a language and code generation framework for heterogeneous targets. *In*: . *[S.l.: s.n.]*, 2016. p. 125–135.

HARVEY, R.; XU, Y.; QU, Z.; NAMERIKAWA, T. Dissipativity-based design of local and wide-area DER controls for large-scale power systems with high penetration of renewables. *In*: **2017 IEEE Conference on Control Technology and Applications (CCTA)**. *[S.l.: s.n.]*, 2017. p. 2180–2187.

HÄSTBACKA, David; VEPSÄLÄINEN, Timo; KUIKKA, Seppo. Model-driven development of industrial process control applications. **Journal of Systems and Software**, v. 84, n. 7, p. 1100 – 1113, 2011. ISSN 0164-1212. Available at: http://www.sciencedirect.com/science/article/pii/S0164121211000458.

HUNDT, Robert. Hp caliper: A framework for performance analysis tools. **IEEE Concurrency**, IEEE Educational Activities Department, USA, v. 8, n. 4, p. 64–71, oct 2000. ISSN 1092-3063.

IEC/IEEE. IEC/IEEE guide for computer-based control for hydroelectric power plant automation. **IEC 62270 Edition 2.0 2013-09 IEEE Std 1249**, p. 1–83, Sept 2013.

INDRUSIAK, L.; GLESNER, M. An actor-oriented model-based design flow for systems-on-chip. *In*: **MBEES - Workshop of Model-Based Development of Embedded Systems**. TU Braunschweig, Germany: *[s.n.]*, 2006. p. 65–73.

JALEELI, N.; VANSLYCK, L.S.; EWART, D.N.; FINK, L.H.; HOFFMANN, A.G. Understanding automatic generation control. **IEEE Transactions on Power Systems**, v. 7, n. 3, p. 1106–1122, 1992.

KERNIGHAN, B. W.; RITCHIE, D. M. **The C Programming Language**. USA: Prentice-Hall, Inc., 1978. ISBN 0131101633.

KLEMPNER, Geoff; KERSZENBAUM, Isidor. **Operation and Maintenance of Large Turbo-Generators**. *[S.l.]*: John Wiley & Sons, 2004. ISBN 0-471-61447-5.

KOSOW, I.L. **Electric Machinery And Transformers 2Nd Ed.** *[S.l.]*: Prentice-Hall, 2009. ISBN 9788120307759.

KUIJLAARS, Ivo. **SCADA Lifecycle Management**. 2015. Seminar presentation at The 13th International Workshop on Electric Power Control Centers.

KUNDUR, P.; BALU, N.J.; LAUBY, M.G. **Power system stability and control**. *[S.l.]*: McGraw-Hill, 1994. (EPRI power system engineering series). ISBN 9780070359581.

LEE, Edward; MESSERSCHMITT, David. Static scheduling of synchronous data flow programs for digital signal processing. **Computers, IEEE Transactions on**, C-36, p. 24 – 35, 02 1987.

LEE, Edward; NEUENDORFFER, Stephen. **MoML - A Modeling Markup Language in XML - Version 0.4**. *[S.l.]*, 2000. 1-14 p. Available at: http://ptolemy.eecs.berkeley.edu/publications/papers/00/moml/.

LEE, Edward; NEUENDORFFER, Stephen; WIRTHLINT, Michael. Actor-oriented design of embedded hardware and software systems. **Journal of Circuits, Systems and Computers**, v. 12, 08 2002.

LEE, E.A.; SESHIA, S.A. **Introduction to Embedded Systems: A Cyber-physical Systems Approach**. *[S.l.: s.n.]*, 2011. (Electrical Engineering & Computer Sciences). ISBN 9780557708574.

LEE, Edward A. Model-driven development - from object-oriented design to actor-oriented design. *In*: **Extended abstract of an invited presentation at Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (a.k.a. The Monterey Workshop)**. Chicago: *[s.n.]*, 2003.

LEE, Edward A. Heterogeneous modeling. *In*: PTOLEMAEUS, Claudius (Ed.). **System Design, Modeling, and Simulation using Ptolemy II**. Ptolemy.org, 2014. Available at: http://ptolemy.org/books/Systems.

LEE, Edward A. Fundamental limits of cyber-physical systems modeling. **ACM Trans. Cyber-Phys. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 1, n. 1, Nov. 2016. ISSN 2378-962X. Available at: https://doi.org/10.1145/2912149.

LEE, Edward Ashford. **Plato and the Nerd: The Creative Partnership of Humans and Technology**. Cambridge, Massachusetts 02142: The MIT Press, 2018. ISBN 0262536420.

LEE, Edward A.; NEUENDORFFER, Stephen; ZHOU, Gang. Dataflow. *In*: PTOLEMAEUS, Claudius (Ed.). **System Design, Modeling, and Simulation using Ptolemy II**. Ptolemy.org, 2014. Available at: http://ptolemy.org/books/Systems.

LEITE, Marcela; WEHRMEISTER, Marco Aurélio. System-level design based on uml/marte for fpga-based embedded real-time systems. **Design Automation for Embedded Systems**, Kluwer Academic Publishers, USA, v. 20, n. 2, p. 127–153, Jun. 2016. ISSN 0929-5585. Available at: https://doi.org/10.1007/s10617-016-9172-6.

LOPEZ, R.; MOORE, A.; GILLERMAN, J. A model-driven approach to smart substation automation and integration for Comision Federal de Electricidad. *In*: **IEEE PES T D 2010**. *[S.l.: s.n.]*, 2010. p. 1–8. ISSN 2160-8555.

LOU, G.; GU, W.; WANG, J.; SHENG, W.; SUN, L. Optimal design for distributed secondary voltage control in islanded microgrids: Communication topology and controller. **IEEE Transactions on Power Systems**, v. 34, n. 2, p. 968–981, March 2019. ISSN 0885-8950.

MARTÍNEZ, J.; KJÆR, P. C.; RODRIGUEZ, P.; TEODORESCU, R. Design and analysis of a slope voltage control for a DFIG wind power plant. **IEEE Transactions on Energy Conversion**, v. 27, n. 1, p. 11–20, March 2012. ISSN 0885-8969.

MENDES, Marcos Fonseca. **Proposta de metodologia e de modelo para modernizações de sistemas de automação de unidades geradoras hidráulicas de grande porte**. May 2011. 259 p. Phd Thesis (PhD Thesis) — Escola Politécnica, Universidade de São Paulo, May 2011.

MOIN, Armin. Data analytics and machine learning methods, techniques and tool for model-driven engineering of smart iot services. *In*: **2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)**. *[S.l.: s.n.]*, 2021. p. 287–292.

MUKHERJEE, Biswarup; VANFRETTI, Luigi. Modeling of PMU-Based automatic re-synchronization controls for DER generators in power distribution networks using Modelica and the OpenIPSL. *In*: **Proceedings of the 13th International Modelica Conference**. Regensburg, Germany: *[s.n.]*, 2019. p. 607–616.

NAZARI, Masoud H.; WANG, Le Yi; GRIJALVA, Santiago; EGERSTEDT, Magnus. Communication-failure-resilient distributed frequency control in smart grids: Part I: Architecture and distributed algorithms. **IEEE Transactions on Power Systems**, v. 35, n. 2, p. 1317–1326, 2020.

NAZARI, Masoud H.; WANG, Le Yi; GRIJALVA, Santiago; EGERSTEDT, Magnus. Communication-failure-resilient distributed frequency control in smart grids: Part II: Algorithmic implementation and system simulations. **IEEE Transactions on Power Systems**, v. 35, n. 4, p. 3192–3202, 2020.

NEIS, P.; SILVA, R. J. G. C.; BASTOS, A. A. Esquema de redução de geração para eliminação de sobrecarga nos autotransformadores 525/765 kv da subestação Foz do Iguaçu através do CAG de Itaipu. *In*: **XI Seminário Técnico de Proteção e Controle**. Florianópolis, SC: *[s.n.]*, 2012.

NEIS, P.; TOCHETTO, A. P.; RAMíREZ, R. J. G.; COSTA, C. H. C.; WEHRMEISTER, M. A. Integração do esquema de redução de geração em Itaipu 60 Hz com o novo SEP do sistema de transmissão em 765 kV: uma abordagem de engenharia guiada por modelos. *In*: **XVII EDAO - Encontro para Debates de Assuntos de Operação**. São Paulo - SP: *[s.n.]*, 2022. p. 1–9.

NEIS, P.; TUFAILE, R. B. R.; FAVORETO, R. S.; SILVA, R. J. G. C.; RIBEIRO, J. R. Maximização da capacidade de escoamento de energia de Itaipu através de sistemas especiais de

proteção. *In*: **XII EDAO - Encontro para Debates de Assuntos de Operação**. Brasília, DF: *[s.n.]*, 2012.

NEIS, P.; WEHRMEISTER, M. A.; MENDES, M. F. Model driven software engineering of power systems applications: Literature review and trends. **IEEE Access**, v. 7, p. 177761–177773, 2019. ISSN 2169-3536.

NEIS, Paulo; WEHRMEISTER, Marco Aurelio; MENDES, Marcos Fonseca; PESENTE, Jonas Roberto. Applying a model-driven approach to the development of power plant SCADA/EMS software. **International Journal of Electrical Power & Energy Systems**, v. 153, p. 109336, 2023. ISSN 0142-0615.

NORDSTROM, Greg; SZTIPANOVITS, Janos; KARSAI, Gabor; LEDECZI, Akos. Metamodeling - rapid design and evolution of domain-specific modeling environments. *In*: **Proceedings of the IEEE ECBS'99 Conference**. Nashville, Tennessee: *[s.n.]*, 1999. p. 68–74.

OLIVEIRA, R. A.; PESENTE, J. R.; SILVA, R. J. G. C. da; NEIS, P.; OTTO, R. B.; RAMOS, R. A. Field experience and recommendations with parameter re-tuning of the load sharing control loops at the Itaipu power plant. *In*: **2017 IEEE Power Energy Society General Meeting**. *[S.l.: s.n.]*, 2017. p. 1–5.

ONS - Operador Nacional do Sistema Elétrico. **Manual de Procedimentos da Operação - Módulo 5 - Submódulo 5.12: Esquemas Especiais da Interligação Sul / Sudeste, Rev. 61**. 2022. Accessed: 2022-October-07. Available at: http://www.ons.org.br/%2FMPO%2FDocumento%20Normativo%2F3.%20Instru%C3%A7%C3%B5es%20de%20Opera%C3%A7%C3%A3o%20-%20SM%205.12%2F3.1.%20Controle%20da%20Transmiss%C3%A3o%2F3.1.2.%20Esquemas%20Especiais%2F3.1.2.1.%20Interliga%C3%A7%C3%A3o%20entre%20Regi%C3%B5es%2FIO-EE.SSE_Rev.61.pdf.

PAZ, Andrés; BOUSSAIDI, Ghizlane El. Breesse: Bridging EMF, Simulink and Stateflow for model-based design of safety-critical systems. *In*: ____. **Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings**. New York, NY, USA: Association for Computing Machinery, 2020. ISBN 9781450381352. Available at: https://doi.org/10.1145/3417990.3421408.

PORTUGAL, Paulo Max Maciel; PERNAS, Renato Weingartner; AMARAL, Lucas Vernot; SILVA, Renata Ribeiro; ABBOUD, Ricardo; CERNEV, Rafael; CABRAL, Marcos. Case study: Modern RAS applied to Furnas 765 kV transmission corridor improves Itaipu power plant and Brazilian power system stability. *In*: **49th Annual Western Protective Relay Conference**. Spokane, WA: *[s.n.]*, 2022. p. 1–12.

ROBERT, G.; HURTADO, D. Optimal design of reactive power PI regulator for hydro power plants. *In*: **2008 IEEE International Conference on Control Applications**. *[S.l.: s.n.]*, 2008. p. 775–780. ISSN 1085-1992.

ROHJANS, S.; PIECH, K.; USLAR, M.; CABADI, J. CIMbaT - automated generation of CIM-based OPC UA-address spaces. *In*: **2011 IEEE International Conference on Smart Grid Communications (SmartGridComm)**. *[S.l.: s.n.]*, 2011. p. 416–421.

SANTOS, Fernando; NUNES, Ingrid; BAZZAN, Ana L.C. Quantitatively assessing the benefits of model-driven development in agent-based modeling and simulation. **Simulation Modelling Practice and Theory**, v. 104, p. 102126, 2020. ISSN 1569-190X.

SCHMIDT, Douglas C. Guest editor's introduction: Model-driven engineering. **Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 39, n. 2, p. 25–31, Feb. 2006. ISSN 0018-9162. Available at: https://doi.org/10.1109/MC.2006.58.

SELIC, B. The pragmatics of model-driven development. **IEEE Software**, v. 20, n. 5, p. 19–25, Sep. 2003. ISSN 0740-7459.

SHAYEGHI, Hossein; JALIL, AREF. Hybrid fuzzy LFC design by GA in a deregulated power system. *In*: **Proceedings of the 6th WSEAS International Conference on Applications of Electrical Engineering**. *[S.l.: s.n.]*, 2007. p. 77–82.

SHENG, G.; LIU, Y.; DUAN, D.; ZENG, Y.; JIANG, X. Secondary voltage regulation based on wide area network. *In*: **2009 IEEE Power Energy Society General Meeting**. *[S.l.: s.n.]*, 2009. p. 1–7. ISSN 1932-5517.

SKOPP, Allen; VARADAN, Srinivas. ABB information & energy management systems support Mexico's growing demand for electric power. **ABB Review**, p. 35–41, 01 2000.

SMYTH, Neil; Davis II, John S.; FENG, Thomas Huining; GOEL, Mudit; LEE, Edward A.; PARKS, Thomas M.; ZHAO, Yang. Process networks and rendezvous. *In*: PTOLEMAEUS, Claudius (Ed.). **System Design, Modeling, and Simulation using Ptolemy II**. Ptolemy.org, 2014. Available at: http://ptolemy.org/books/Systems.

SOMMERVILLE, I. **Software Engineering**. *[S.l.]*: Pearson, 2011. (International Computer Science Series). ISBN 9780137053469.

STIFTER, M.; WIDL, E.; ANDRÉN, F.; ELSHEIKH, A.; STRASSER, T.; PALENSKY, P. Co-simulation of components, controls and power systems based on open source software. *In*: **2013 IEEE Power Energy Society General Meeting**. *[S.l.: s.n.]*, 2013. p. 1–5. ISSN 1932-5517.

STRASSER, Thomas; JONG, Erik; SOSNINA, Maria. **European Guide to Power System Testing: The ERIGrid Holistic Approach for Evaluating Complex Smart Grid Configurations**. *[S.l.: s.n.]*, 2020. ISBN 978-3-030-42273-8.

STRASSER, T.; ROOKER, M.; HEGNY, I.; WENGER, M.; ZOITL, A.; FERRARINI, L.; DEDE, A.; COLLA, M. A research roadmap for model-driven design of embedded systems for automation components. *In*: **2009 7th IEEE International Conference on Industrial Informatics**. *[S.l.: s.n.]*, 2009. p. 564–569. ISSN 1935-4576.

STRASSER, T.; STIFTER, M.; ANDRÉN, F.; PALENSKY, P. Co-simulation training platform for smart grids. **IEEE Transactions on Power Systems**, v. 29, n. 4, p. 1989–1997, July 2014. ISSN 0885-8950.

SUBRAMANYA, K. **Hydraulic Machines:**. *[S.l.]*: Tata McGraw-Hill Education, 2013. ISBN 9789332900981.

SULLIGOI, G.; CHIANDONE, M.; ARCIDIACONO, V. NewSART automatic voltage and reactive power regulator for secondary voltage regulation: Design and application. *In*: **2011 IEEE Power and Energy Society General Meeting**. *[S.l.: s.n.]*, 2011. p. 1–7. ISSN 1932-5517.

SUN, H.; GUO, Q.; ZHANG, B.; WU, W.; TONG, J. Development and applications of system-wide automatic voltage control system in China. *In*: **2009 IEEE Power Energy Society General Meeting**. *[S.l.: s.n.]*, 2009. p. 1–5. ISSN 1932-5517.

SÜSS, J. G.; POP, A.; FRITZSON, P.; WILDMAN, L. Towards integrated model-driven testing of SCADA systems using the Eclipse Modeling Framework and Modelica. *In*: **19th Australian Conference on Software Engineering (aswec 2008)**. *[S.l.: s.n.]*, 2008. p. 149–159. ISSN 1530-0803.

The Object Management Group. OMG Unified Modeling Language (OMG UML), superstructure, v2.1.2. **OMG Available Specification**, November 2007. Available at: https://www.omg.org/spec/UML/2.1.2/Superstructure/PDF.

USLAR, Mathias; ROHJANS, Sebastian; NEUREITER, Christian; ANDRÉN, Filip Pröstl; VELASQUEZ, Jorge; STEINBRINK, Cornelius; EFTHYMIOU, Venizelos; MIGLIAVACCA, Gianluigi; HORSMANHEIMO, Seppo; BRUNNER, Helfried; STRASSER, Thomas I. Applying the smart grid architecture model for designing and validating system-of-systems in the power and energy domain: A european perspective. **Energies**, v. 12, n. 2, 2019. ISSN 1996-1073.

VAN-SLYKE, Doug. **SCADA: The Heart of an Energy Management System**. 2015. Seminar presentation, IEEE IAS-PES Chapter, Southern Alberta Section.

VIRMANI, Sudhir; SAVULESCU, Savu. The real-time and study-mode data environment in modern scada/ems. *In*: SAVULESCU, Savu C. (Ed.). **Real Time Stability Assessment in Modern Power System Control Centers**. *[S.l.]*: John Wiley & Sons, Ltd, 2008. p. 1–21. ISBN 9780470233306.

VOELTER, Markus; KOLB, Bernd; BIRKEN, Klaus; TOMASSETTI, Federico; ALFF, Patrick; WIART, Laurent; WORTMANN, Andreas; NORDMANN, Arne. Using language workbenches and domain-specific languages for safety-critical software development. **Softw. Syst. Model.**, Springer-Verlag, Berlin, Heidelberg, v. 18, n. 4, p. 2507–2530, aug 2019. ISSN 1619-1366.

VOINOV, A.; YANG, C.; VYATKIN, V. Automatic generation of function block systems implementing HMI for energy distribution automation. *In*: **2017 IEEE 15th International Conference on Industrial Informatics (INDIN)**. *[S.l.: s.n.]*, 2017. p. 706–713. ISSN 2378-363X.

WANG, N.; CHEUNG, R.; WU, G.; NACCARINO, J.; CASTLE, J. Simulation of the New York Power Pool for dispatcher training. **IEEE Transactions on Power Systems**, v. 9, n. 4, p. 2063–2072, 1994.

WANG, Yu; NGUYEN, Tung Lam; XU, Yan; LI, Zhengmao; TRAN, Quoc-Tuan; CAIRE, Raphael. Cyber-physical design and implementation of distributed event-triggered secondary control in islanded microgrids. **IEEE Transactions on Industry Applications**, v. 55, n. 6, p. 5631–5642, 2019.

WEHRMEISTER, Marco Aurélio. **An Aspect-Oriented Model-Driven Engineering Approach for Distributed Embedded Real-Time Systems**. 2009. 206 p. Phd Thesis (PhD Thesis) — Universidade Federal do Rio Grande do Sul, 2009.

WEHRMEISTER, Marco Aurélio; de Freitas, Edison Pignaton; BINOTTO, Alécio Pedro Delazari; PEREIRA, Carlos Eduardo. Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems. **Mechatronics**, v. 24, n. 7, p. 844–865, 2014. ISSN 0957-4158. 1. Model-Based Mechatronic System Design 2. Model Based Engineering.

WEHRMEISTER, Marco Aurélio; FREITAS, Edison Pignaton de; PEREIRA, Carlos Eduardo; WAGNER, Flavio Rech. An aspect-oriented approach for dealing with non-functional requirements in a model-driven development of distributed embedded real-time systems. *In*: **Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing**. Washington: IEEE Computer Society, 2007. ISBN 0769527655.

WEHRMEISTER, Marco Aurélio; FREITAS, Edison Pignaton de; PEREIRA, Carlos Eduardo; RAMMIG, Franz Joseph. Combining aspects-oriented concepts with model-driven techniques in the design of distributed embedded real-time systems. *In*: **Proceedings of the Work-in-Progress Session of the 19th Euromicro Conference on Real-Time Systems**. Sigapura: National University of Singapore, 2007.

WEHRMEISTER, Marco Aurélio; FREITAS, Edison Pignaton de; BINOTTO, Alécio Pedro Delazari; PEREIRA, Carlos Eduardo. Combining aspects and object-orientation in

model-driven engineering for distributed industrial mechatronics systems. **Mechatronics**, v. 24, n. 7, p. 844 – 865, 2014. ISSN 0957-4158. Available at: http://www.sciencedirect.com/science/article/pii/S0957415813002420.

WEHRMEISTER, M. A.; PEREIRA, C. E.; RAMMIG, F. J. Aspect-oriented model-driven engineering for embedded systems applied to automation systems. **IEEE Transactions on Industrial Informatics**, v. 9, n. 4, p. 2373–2386, 2013.

YANG, C.; DUBININ, V.; VYATKIN, V. Ontology driven approach to generate distributed automation control from substation automation design. **IEEE Transactions on Industrial Informatics**, v. 13, n. 2, p. 668–679, April 2017. ISSN 1551-3203.

YANG, Chen-Wei; DUBININ, Victor; VYATKIN, Valeriy. Automatic generation of control flow from requirements for distributed smart grid automation control. **IEEE Transactions on Industrial Informatics**, v. 16, n. 1, p. 403–413, 2020.

YANG, C. W.; VYATKIN, V. On requirements-driven design of distributed smart grid automation control. *In*: **2017 IEEE 15th International Conference on Industrial Informatics (INDIN)**. *[S.l.: s.n.]*, 2017. p. 738–745.

ZANABRIA, C.; ANDRÉN, F. P.; KATHAN, J.; STRASSER, T. Towards an integrated development of control applications for multi-functional energy storages. *In*: **2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)**. *[S.l.: s.n.]*, 2016. p. 1–4.

ZANABRIA, Claudia; ANDRÉN, Filip Pröstl; STRASSER, Thomas I.; KASTNER, Wolfgang. A model-driven and ontology-based engineering approach for smart grid automation applications. *In*: **IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society**. *[S.l.: s.n.]*, 2019. v. 1, p. 6635–6641.

ZANABRIA, Claudia; TAYYEBI, Ali; ANDRÉN, Filip Pröstl; KATHAN, Johannes; STRASSER, Thomas. Engineering support for handling controller conflicts in energy storage systems applications. **Energies**, v. 10, n. 10, 2017. ISSN 1996-1073. Available at: https://www.mdpi.com/1996-1073/10/10/1595.

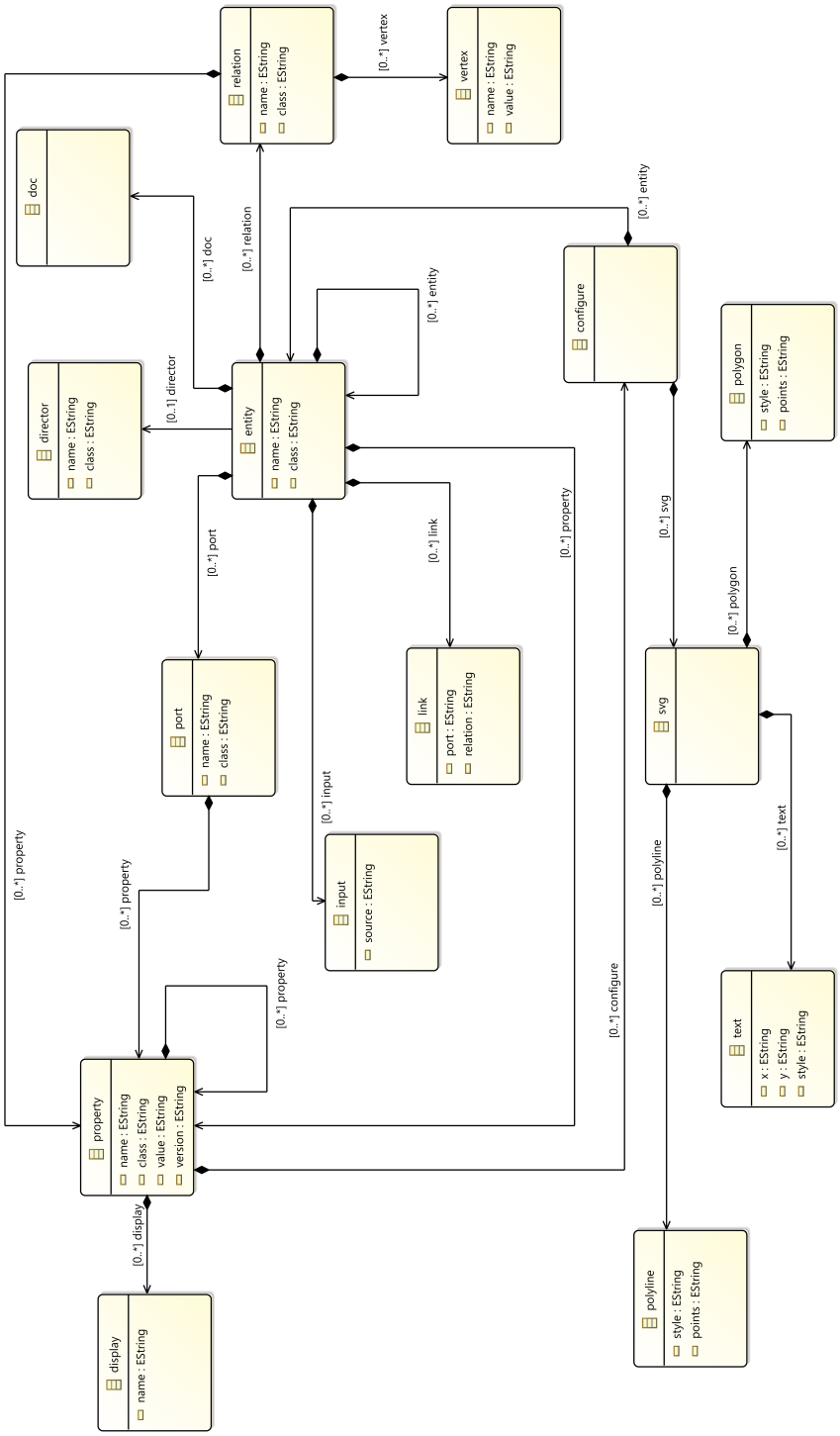
ZHABELOVA, G.; YANG, C. W.; PATIL, S.; PANG, C.; YAN, J.; SHALYTO, A.; VYATKIN, V. Cyber-physical components for heterogeneous modelling, validation and implementation of smart grid intelligence. *In*: **2014 12th IEEE International Conference on Industrial Informatics (INDIN)**. *[S.l.: s.n.]*, 2014. p. 411–417. ISSN 1935-4576.

ZHOU, Gang; LEUNG, Man-Kit; LEE, Edward A. **A Code Generation Framework for Actor-Oriented Models with Partial Evaluation**. EECS Department, University of California, Berkeley, 2007. Available at: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-29.html.

APPENDIX

APPENDIX A – EMSML METAMODEL UML CLASS DIAGRAM

Figure 65 – EMSML full metamodel.



Source: The author.

APPENDIX B – THE TOY JBVRC CONTROLLER XMI FILE, GENERATED CODE AND EXAMPLE PROGRAM

Listing B.1 – A toy application modeled with EMSML concrete syntax.

```
1 <?xml version="1.0" encoding="ASCII"?><EMSML:entity xmlns:EMSML="http://EMSML.ecore.gss.
      itaipu/EMSML" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/
      XMLSchema-instance" class="ptolemy.actor.TypedCompositeActor" name="JBVRC_toy"
      xmi:version="2.0" xsi:schemaLocation="http://EMSML.ecore.gss.itaipu/EMSML file:/Users/
      pneis/eclipse-workspace/itaipu.gss.ecore.EMSML/model/EMSML.ecore">
2     <property class="ptolemy.kernel.attributes.VersionAttribute" name="_createdBy" value="
          11.0.1"> </property>
3 <!-- The toy model runs under SDF MoC, for a configurable number of iterations: -->
4   <property class="ptolemy.domains.sdf.kernel.SDFDirector" name="SDF Director">
5         <property class="ptolemy.data.expr.Parameter" name="iterations" value="10"> </
              property>
6     </property>
7 <!-- The model contains a composite actor names JBVRC: -->
8     <entity class="ptolemy.actor.TypedCompositeActor" name="JBVRC">
9         <property class="ptolemy.domains.sdf.kernel.SDFDirector" name="SDF Director">
10            <property class="ptolemy.data.expr.Parameter" name="iterations" value="AUTO">
                  </property>
11        </property>
12 <!-- The composite JBVRC has ports: -->
13        <port class="ptolemy.actor.TypedIOPort" name="In Voltage">
14            <property name="input"/>   </port>
15 ...
16 <!-- The composite JBVRC contains other actors: -->
17        <entity class="ptolemy.actor.lib.AddSubtract" name="CalcError"> </entity>
18        <entity class="ptolemy.actor.lib.AddSubtract" name="AddSubtract"> </entity>
19        <entity class="ptolemy.actor.lib.Scale" name="Kp">  </entity>
20        <entity class="ptolemy.actor.lib.IIR" name="IIR Transf Fun">
21            <property class="ptolemy.data.expr.Parameter" name="numerator" value="{0.33,
                  0.33}"> </property>
22            <property class="ptolemy.data.expr.Parameter" name="denominator" value="{1,
                  -0.33}"> </property>
23        </entity>
24 <!-- The composite contains relations: -->
25        <relation class="ptolemy.actor.TypedIORelation" name="relation3"></relation>
26        <relation class="ptolemy.actor.TypedIORelation" name="relation4"></relation>
27 ...
28 <!-- The composite contains links connecting ports to relations: -->
29        <link port="In Voltage" relation="relation4"/>
30        <link port="Unit Setp" relation="relation2"/>
31 ...
32    </entity> <!-- End of JBVRC composite actor -->
33 <!-- Other actors in the model: -->
34    <entity class="ptolemy.actor.lib.Const" name="Const">
```

```
35          <property class="ptolemy.data.expr.Parameter" name="value" value="1"> </property>
36      </entity>
37      <entity class="ptolemy.actor.lib.gui.Display" name="Display">  </entity>
38  ...
39  <!-- Relations for the outer model: -->
40      <relation class="ptolemy.actor.TypedIORelation" name="relation2"> </relation>
41      <relation class="ptolemy.actor.TypedIORelation" name="relation"> </relation>
42      <relation class="ptolemy.actor.TypedIORelation" name="relation3"> </relation>
43  <!-- Links connecting relations for the outer model: -->
44      <link port="JBVRC.In Voltage" relation="relation3"/>
45      <link port="JBVRC.Unit Setp" relation="relation"/>
46      <link port="JBVRC.Ref Voltage" relation="relation2"/>
47      <link port="JBVRC.Avg Unit Reac Pow" relation="relation2"/>
48      <link port="Const.output" relation="relation2"/>
49      <link port="Display.input" relation="relation"/>
50      <link port="Sequence.output" relation="relation3"/>
51  </EMSML:entity>
```

Listing B.2 – Output header file for the toy JBVRC.

```
1  /*********************************************************************/
2  /* This module is proprietary to:                                    */
3  /*      Itaipu - OP.DT - GSS                                         */
4  /* Auto generated with Eclipse EMF / Acceleo - DO NOT EDIT           */
5  /*********************************************************************/
6
7  #include "CompositeEntity.h"
8  #include "TypedIORelation.h"
9  #include "AddSubtract.h"
10 #include "Scale.h"
11 #include "IIR.h"
12
13 class JBVRC;
14
15
16 class JBVRC : public CompositeEntity {
17 public:
18     JBVRC(std::string sEntityName, Entity* pContainerEntity=NULL);
19     ~JBVRC();
20     virtual bool initialize();
21
22 private:
23     AddSubtract* pCalcError{NULL};
24     AddSubtract* pAddSubtract{NULL};
25     Scale* pKp{NULL};
26     IIR* pIIR_Transf_Fun{NULL};
27
28     TypedIORelation relation3;
29     TypedIORelation relation4;
```

```
30      TypedIORelation relation;
31      TypedIORelation relation5;
32      TypedIORelation relation10;
33      TypedIORelation relation6;
34      TypedIORelation relation2;
35
36  };
```

Listing B.3 – Output implementation file for the toy JBVRC.

```
1  /*************************************************************************/
2  /* This module is proprietary to:                                        */
3  /*      Itaipu - OP.DT - GSS                                             */
4  /* Auto generated with Eclipse EMF / Acceleo - DO NOT EDIT               */
5  /*************************************************************************/
6  #include "Port.h"
7  #include "Token.h"
8  // These two are needed because abs() from std C differs from std::abs
9  #include <cmath>
10 using namespace std;
11
12 #include "JBVRC.h"
13
14 /*********************************************************
15  * Follows methods implemented for class JBVRC
16 *********************************************************/
17 JBVRC::JBVRC(std::string sEntityName, Entity* pContainerEntity) : CompositeEntity(
       sEntityName, pContainerEntity)
18 {
19 }
20
21 JBVRC::~JBVRC()
22 {
23      delete pCalcError;
24      delete pAddSubtract;
25      delete pKp;
26      delete pIIR_Transf_Fun;
27 }
28
29 bool JBVRC::initialize()
30 {
31      bool bInitStatus = true;
32
33      /* Initialize composite and instantiate contained entities */
34      // Contained entities:
35      pCalcError = new AddSubtract("CalcError", this);
36      addComponentEntity(pCalcError);
37
38      pAddSubtract = new AddSubtract("AddSubtract", this);
```

```
39      addComponentEntity(pAddSubtract);
40
41      pKp = new Scale("Kp", this);
42      addComponentEntity(pKp);
43
44      pIIR_Transf_Fun = new IIR("IIR_Transf_Fun", this);
45      addComponentEntity(pIIR_Transf_Fun);
46
47      this->setParameter("_flipPortsHorizontal", "false");
48      this->setParameter("_flipPortsVertical", "false");
49      pKp->setParameter("factor", "1");
50      pIIR_Transf_Fun->setParameter("numerator", "{0.33, 0.33}");
51      pIIR_Transf_Fun->setParameter("denominator", "{1, -0.33}");
52      bInitStatus &= pCalcError->initialize();
53      bInitStatus &= pAddSubtract->initialize();
54      bInitStatus &= pKp->initialize();
55      bInitStatus &= pIIR_Transf_Fun->initialize();
56
57      /* Instantiate ports */
58      newPort( "In_Voltage", 'I', Port::TypedIOPort);
59      newPort( "Unit_Setp", 'O');
60      newPort( "Ref_Voltage", 'I', Port::TypedIOPort);
61      newPort( "Avg_Unit_Reac_Pow", 'I', Port::TypedIOPort);
62
63      /* Initialize relations */
64      relation3.setName("relation3");
65      relation4.setName("relation4");
66      relation.setName("relation");
67      relation5.setName("relation5");
68      relation10.setName("relation10");
69      relation6.setName("relation6");
70      relation2.setName("relation2");
71
72      bInitStatus &= relation4.connect(getPort("In_Voltage"), true);
73      bInitStatus &= relation2.connect(getPort("Unit_Setp"), true);
74      bInitStatus &= relation3.connect(getPort("Ref_Voltage"), true);
75      bInitStatus &= relation6.connect(getPort("Avg_Unit_Reac_Pow"), true);
76      bInitStatus &= relation3.connect(pCalcError->getPort("plus"));
77      bInitStatus &= relation4.connect(pCalcError->getPort("minus"));
78      bInitStatus &= relation5.connect(pCalcError->getPort("output"));
79      bInitStatus &= relation10.connect(pAddSubtract->getPort("plus"));
80      bInitStatus &= relation6.connect(pAddSubtract->getPort("plus"));
81      bInitStatus &= relation.connect(pAddSubtract->getPort("output"));
82      bInitStatus &= relation.connect(pKp->getPort("input"));
83      bInitStatus &= relation2.connect(pKp->getPort("output"));
84      bInitStatus &= relation5.connect(pIIR_Transf_Fun->getPort("input"));
85      bInitStatus &= relation10.connect(pIIR_Transf_Fun->getPort("output"));
86
87
```

```
88      setInitialized(bInitStatus);
89      return bInitStatus;
90  }
```

Listing B.4 – C++ main module invoking the toy JBVRC controller.

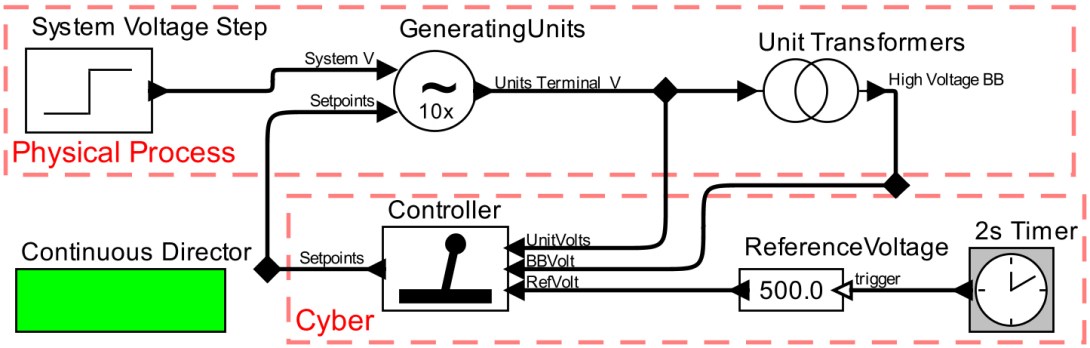
```
1  #include <iostream>
2  #include <iomanip>
3  #include "JBVRC.h"
4  #include "SdfSeqScheduler.h"
5  #include "Token.h"
6  #include "Port.h"
7
8  int main(int argc, char* argv[])
9  {
10     int iRetVal = 0;
11     bool bContinue = true;
12     JBVRC* pJ = new JBVRC("JBVRC");
13     TypedIORelation inVoltRelat, in2Relat, outRelat;
14     inVoltRelat.setName("inVoltRelat");
15     outRelat.setName("outRelat");
16     in2Relat.setName("in2Relat");
17
18     std::cout << std::setprecision(13);
19     pJ->initialize();
20     // Connect relations to the composite JBVRC's ports:
21     bContinue &= inVoltRelat.connect(pJ->getPort("In_Voltage"));
22     bContinue &= in2Relat.connect(pJ->getPort("Ref_Voltage"));
23     bContinue &= in2Relat.connect(pJ->getPort("Avg_Unit_Reac_Pow"));
24     if( !bContinue ){
25         std::cout << "main(): Failed connecting InPorts...\n";
26         return -1;
27     }
28     // Compute the SDF's schedule previous to main execution loop:
29     SdfSeqScheduler sch;
30     sch.computeSchedule(pJ) ;
31     std::cout << "\n#########################################################\n";
32     Token t(1.0);
33     // Run while input arguments exist in the cmd line:
34     for(int i=1; i<argc; i++){
35         Token tInVol( std::vector<double>(4, std::stod(argv[i])) );
36         inVoltRelat.dispatchToken(tInVol);
37         in2Relat.dispatchToken(t);
38         bool bResult = pJ->fire(); // This is the actual JBVRC iteration
39         if( !bResult ){
40             std::cout << "\nmain(): FAILED firing JBVRC!\n\n";
41         }
42     }
43     // Print all calculated tokens to the console:
```

```
44      Token tOut = pJ->getPort("Unit_Setp")->getToken();
45      while( tOut.isValid() ){
46          std::cout << tOut.print() << ", \n";
47          tOut = pJ->getPort("Unit_Setp")->getToken();
48      }
49      std::cout <<  std::endl;
50      delete(pJ);
51      return iRetVal;
52  }
```

APPENDIX C – HIERARCHICAL COMPOSITION OF THE JBVRC AND PHYSICAL PROCESS

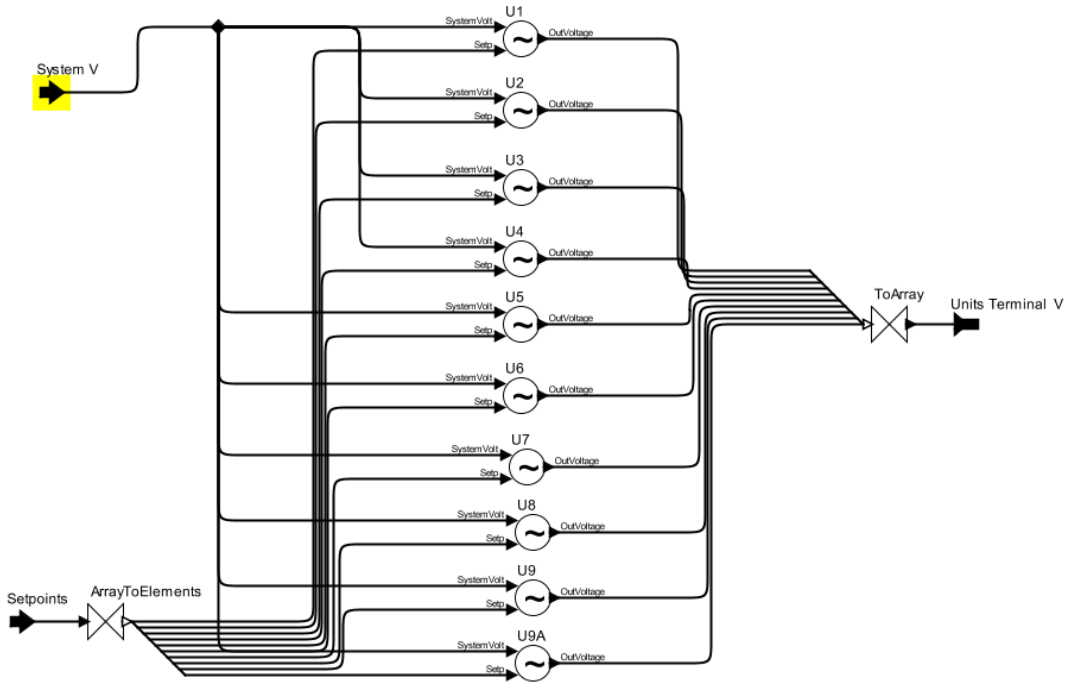
Note: potentiometer discrete transfer function from Figure 70 is $1/(z-1)$, which represents a discrete integrator.

Figure 66 – Model of power plant and secondary voltage controller.



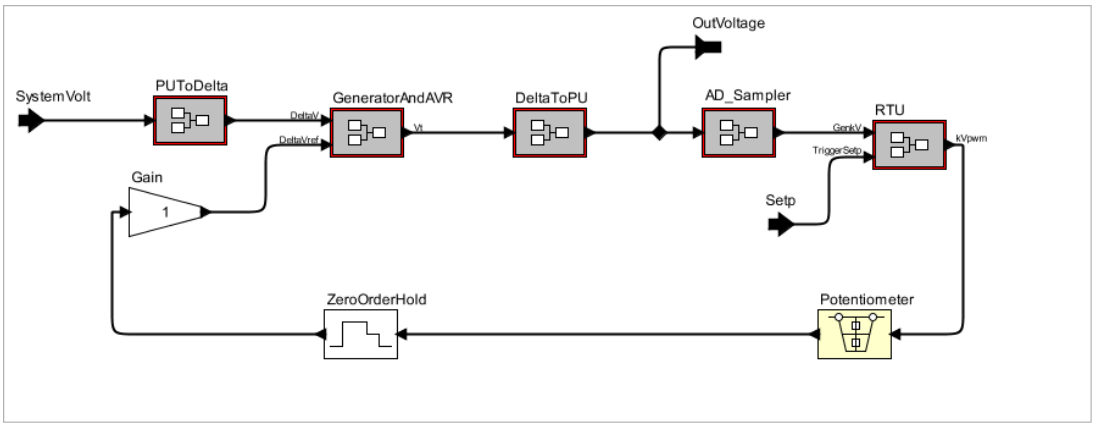
Source: Neis *et al.* (2023).

Figure 67 – Generating Units model.



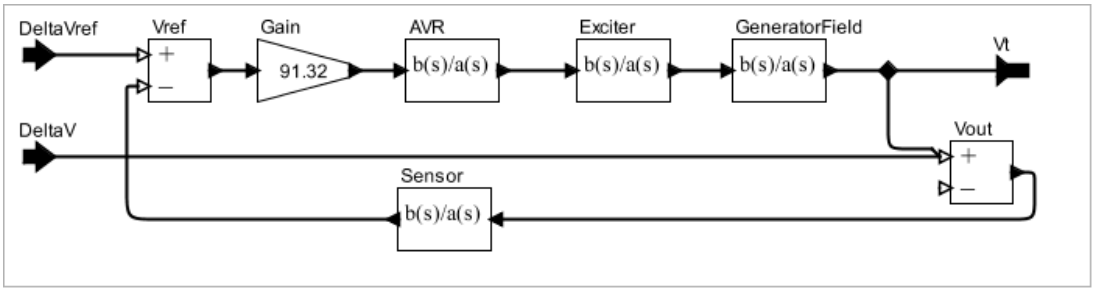
Source: The author.

Figure 68 – A single Generating Unit's voltage control model.



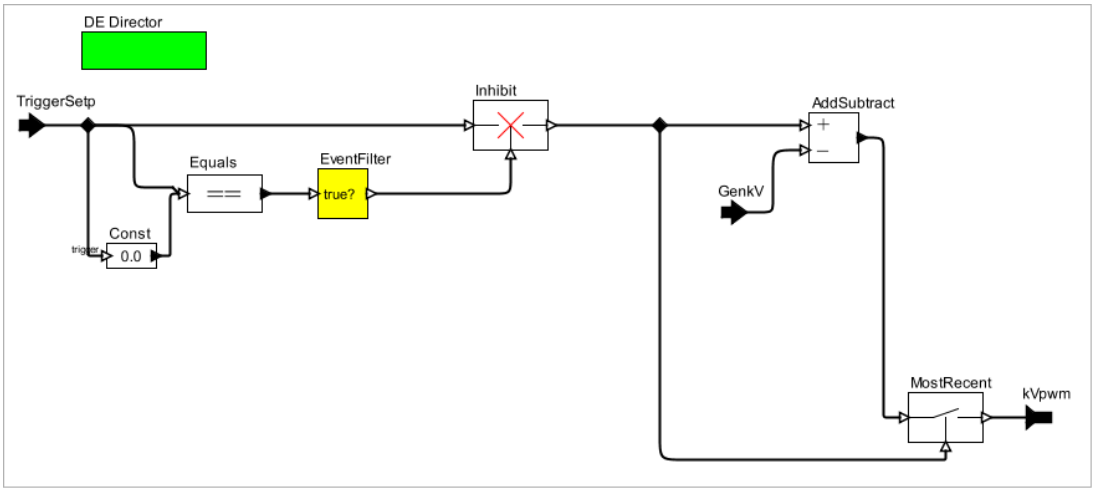
Source: The author.

Figure 69 – Generator and AVR model.



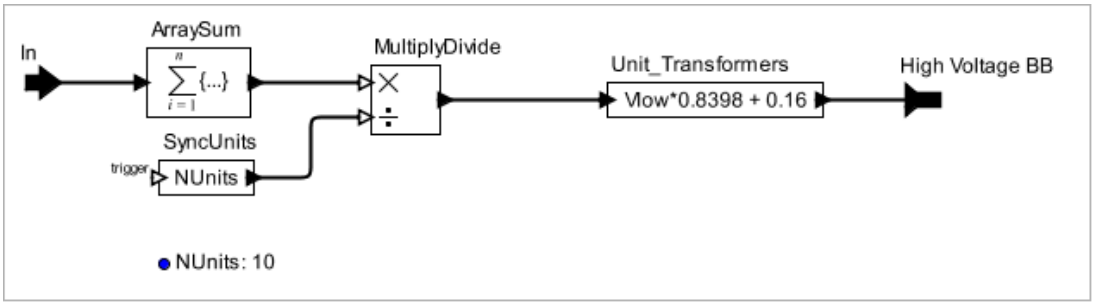
Source: The author.

Figure 70 – Generating Unit's RTU model.



Source: The author.

Figure 71 – Generating Units' step up transformer model.



Source: The author.