Start of a new design method for a competitive Small Modular Reactor (SMR) adaptable to future uses

KTH Master Thesis Report

BAPTISTE MAZURIÉ

Authors

Baptiste Mazurié <mazurie@kth.se>
Engineering Mechanics
KTH Royal Institute of Technology

Place for Project

EDF Lab Chatou, 6 Quai Watier, 78400 Chatou, France

Examiner

Anders Dahlkild <ad@mech.kth.se>
Associate Professor
Teknikringen 8, KTH Royal Institute of Technology

Supervisor

Audrey Jardin <audrey.jardin@edf.fr>

Research engineer

EDF Lab Chatou, 6 Quai Watier, 78400 Chatou, France

Abstract

As cyber-physical systems become increasingly complex, the management and verification of requirements during design is essential. A new language called CRML (Common Requirement Modelling Language) has been created during the European EMBrACE project to formalize realistic dynamical requirements, but a method for representing these requirements and a framework for using them as a design aid must be defined to ease appropriation by engineers. This report therefore presents a draft of a new graphical method for representing system requirements and evaluating different architectures, extending classical Systems Engineering (SE) approaches to the complexity of dynamic physical systems. Once the method completed, the verification and validation process can then be carried out through the simulation of solution models with CRML models. Solution models state how the system will behave while the CRML models state whether this behaviour is compliant with the requirements. The new graphical approach has been applied to two energy systems found in the nuclear industry. This report presents the first promising results as well as some perspectives to consolidate it.

Keywords

Requirement modelling, Cyber-physical systems, Small Modular Reactor (SMR), Systems Engineering, CRML

Abstract

Eftersom cyberfysiska system blir alltmer komplexa är det viktigt att hantera och verifiera kraven under konstruktionen. Ett nytt språk kallat CRML (Common Requirement Modelling Language) har skapats under det europeiska EMBrACE-projektet för att formalisera realistiska dynamiska krav, men en metod för att representera dessa krav och en ram för att använda dem som konstruktionshjälp måste definieras för att underlätta för ingenjörer att använda dem. I denna rapport presenteras därför ett utkast till en ny grafisk metod för att representera systemkrav och utvärdera olika arkitekturer, som utvidgar klassiska systemteknikmetoder till att omfatta komplexiteten hos dynamiska fysiska system. När metoden är färdig kan verifierings- och valideringsprocessen sedan genomföras genom simulering av lösningsmodeller med CRML-modeller. Lösningsmodellerna anger hur systemet kommer att bete sig medan CRML-modellerna anger om detta beteende överensstämmer med kraven. Den nya grafiska metoden har tillämpats på två energisystem som finns inom kärnkraftsindustrin. I denna rapport presenteras de första lovande resultaten samt några perspektiv för att konsolidera dem.

Nyckelord

Modellering av krav, cyberfysiska system, Small Modular Reactor (SMR), systemteknik, CRML

Acknowledgements

I would like to thank EDF R&D, the PRISME department and the P13 team for giving me this opportunity to do a Master Thesis. It allowed me to discover the functioning of EDF as well as the life in a company in a research department. I want to thank especially Audrey Jardin and Daniel Bouskela for helping and guiding me during 6 months. Their support was essential for the success of the Master Thesis.

Finally I would like to thank Anders Dahlkild for being my tutor and examiner at KTH.

Acronyms

CPS	Cyber-Physical Systems
CRML	Common Requirement Modelling Language
ICS	Intermediate Cooling System
EDF	Electricité de France
EMBrACE	Environment for model-based rigorous adaptive co-design and operation of CPS
EPR	European Pressurized Reactor
ETL	Extended Temporal Language
MSR	Moisture Separator Reheater
FORM-L	Formal Requirement Modelling Language
ME-CPS	MultiEnergy Cyber-Physical Systems
NPSH	Net Positive Suction Head
NUWARD	Nuclear Forward
PWR	Pressurised Water Reactor
RISE	Research institutes of Sweden SICS East
SMR	Small Modular Reactor
SE	Systems Engineering
UML	Unified Modelling Language
R&D	Research and Development
I&C	Instrumentation and Control
P&ID	Piping and Instrumentation Diagram

Contents

1 Introduction 1

1.1 Background 1
1.2 Business and technical problem 2
1.3 Goal of the Master Thesis 2
1.4 Document purpose 3
1.5 Benefits, Ethics and Sustainability 3
1.6 Methodology 4
1.7 Stakeholders 4
1.8 Outline 4

2 Framework for the design and verification of complex systems 5

2.1 EDF's background on model-based system engineering 5
2.1.1 Motivation and start of requirements modelling at EDF 5
2.1.2 Development of Formal Requirement Modelling Language (FORM-L), ReqSysPro and ETL 6
2.1.3 Emergence of CRML and current situation 6
2.2 Requirements modelling 7
2.2.1 Requirements definition 7
2.2.2 Formalization of requirements 9
2.3 Verification model: a way to use requirements in a design process 11
2.3.1 Behavioural models 12
2.3.2 Architectural models 12
2.3.3 Requirements model 12
2.3.4 Process for model development 14
2.3.5 Bindings: a way to connect models 14

3 Towards a new graphical method for systems design 17

3.1 Goal of the method 17
3.2 The new graphical method for expressing requirements 18

3.2.1 Main principles 19
3.2.2 Inheritance diagram 21
3.2.3 Internal structure diagram 26
3.2.4 Instance diagrams 30
3.3 Simulation to evaluation requirements generated by the graphical 34

4 Application for the design of SMR 38
4.1 SMR and non-electric applications 38
4.2 Application of the graphical method to the secondary circuit of an SMR . 39
4.2.1 First design of the secondary loop as a “classical” Rankine cycle . 40
4.2.2 Refined design to improve the efficiency of the Rankine cycle . . . 42
4.2.3 Further refinement by defining concrete components and their associated requirements 45

5 Conclusions 47
5.1 Future Work 48
5.2 Final Words 48

References 49

Chapter 1

Introduction

1.1 Background

In order to renew the French electricity production fleet and to address export market, EDF is working on the design of new nuclear reactors. For example, the concept of an European Pressurized Reactor (EPR) is studied at EDF with the development of the EPR (1650 MWe) in Flamanville, France or in Hinkley Point, England as well as the development of a new type of reactor, the Small Modular Reactor (SMR), which is underway with the NUWARD™ project (2x170 MWe). The project NUWARD™ aims to be competitive on the European and international market in particular for replacing ageing coal-fired plants in the 300-400MWe range, supplying remote municipalities and energy-intensive industrial sites, powering grids with limited capacity for large power plants. NUWARD™ competitiveness is based on three levers: (1) simplicity of design, (2) modularity of design and (3) the greatest possible standardisation to benefit from series effects.

The design of a reactor is an extremely complex project involving a multitude of stakeholders from different fields. Indeed, such projects require engineering expertise in various domains: from the reaction of the atoms in the core of the plant, to the production of electricity in the secondary circuit while respecting the different safety and environmental standards and contracting with multiple components providers. Thus, many stakeholders are interacting.

Such characteristics are not unique to the nuclear field and can be found in many other fields of engineering such as the automotive, aviation and aerospace industries. This report proposes a new graphical design method issued from the main Systems Engineering principle which had proven its value particularly in the software and transportation sectors. Even if only nuclear examples are addressed here, report focuses on energy systems at large, and the term

MultiEnergy Cyber-Physical Systems (ME-CPS) can therefore be used in a more general way than just the case of the nuclear application.

1.2 Business and technical problem

ME-CPS projects, and more particularly projects in the nuclear industry, are long-term works with significant funding and risk-taking. Thus, to remain competitive, projects must take into account all the constraints and requirements of the stakeholders and consider the different scenarios of use to find a compromise between the parties.
Moreover, the last few years have shown various problems in the nuclear industry, in particular the loss of knowledge and skills between the teams that built the first reactors in France and those designing the new reactors today. An effort must be made to preserve knowledge, so that in a few years' time, the justifications for the choices made for the SMRs can be restored and understood.

1.3 Goal of the Master Thesis

EDF must act to make its complex projects, and in particular NUWARD™, more competitive. To this end, new engineering methods are being developed to ensure that the initial design of NUWARD™ respects its design pillars: modularity and adaptation to different environments (export projects) or uses (electric and non-electric uses). Ensuring this on the initial design also helps to avoid late reconsideration of choices made during the project which could lead to over-cost and reduce product competitiveness.

Thus, the PRISME department of EDF Research and Development (R&D) is developing new methods of aided design, in particular at system scale (0D/1D scale). One of the department's research axes is to use "systems engineering" techniques to help designing ME-CPS. This work is done in collaboration with other partners in the framework of the European ITEA3 EMBrACE project (notably with Saab, Siemens, the University of Linköping, RISE, etc.). This project aims in particular at the definition of a new language called Common Requirement Modelling Language (CRML), and to promote it as a future standard for extending the "systems engineering" approaches to dynamic physical systems. In practice, the main idea of CRML is to formalize the dynamic and realistic constraints and requirements of the system in order to be able to compute them and to compare, by simulation, the relevance of different design alternatives depending on how well they are compliant with the requirements.

This internship therefore consists in developing a graphical method to facilitate the development of requirements models and its appropriation in an engineering environment.

The aim of the internship is to test this modelling approach (combining system's insights on both its physical and requirements parts) in order to initiate a design methodology that allows:

- the justification of the design choices made at each stage of an engineering project
- the evaluation of the impact of a project modification (standard/market evolution, change of customer's expectations, ...) on the current design to ensure a good coordination and understanding of each stakeholder

1.4 Document purpose

The objective of this report is to summarize the solutions found for initiating the graphical method and to test them to a complete example. This report should enable the reader to understand the basics of the method in its entirety, as well as some perspectives to improve it for a reuse on other cases.

1.5 Benefits, Ethics and Sustainability

The first results presented in this report are directly applicable to the nuclear domain, but it is important to specify that the method has been conceived to be part of a wider framework as studied by [1] and aims at studying ME-CPS at large. It can therefore theoretically be used by many people/industries and thus play a key role in improving globally the design of (energy) systems. Indeed, the aim being to facilitate the representation and modelling of requirements in order to choose the best possible choice between different solution architectures over multiple usage scenarios, this methodology could have a real impact on the management of numerous large engineering projects. This allows engineers to save time on testing complex, repetitive requirements in the case of different architectures/scenarios and ensure a better coordination of the different engineering teams/stakeholders involved.

The new method also allows for a better knowledge tracking and therefore the ability to use feedback from previous designs. This is crucial in the future to understand the choices made in the past but also to make projects cheaper. Indeed, understanding the choices made makes it possible to optimise design choices in the future in order to reduce the excessive margins taken by the accumulation of many stakeholders.

1.6 Methodology

Understanding and assessing the engineering needs of the energy domain in terms of requirements modelling was essential to develop the graphical method. This was done by learning about the requirements of complex projects such as the SMR and by discussing with my tutors at EDF. As it is a new method, no previous example could be found, and creative work had to be done. CRML being also a quite new language with supporting tools still in development, the specifications of the language were very useful to understand what the graphical method had to represent.

1.7 Stakeholders

This master thesis is part of a larger project at EDF R&D implying many stakeholders both internally and externally.
At first, it is in relation with the emergence of the CRML language and therefore of the EMBrACE project. My tutors at EDF guided me towards what was expected from the new graphical method. As the release of CRML specifications is recent and its tool support is still under development, they also helped me to be up to date with the evolution of the CRML syntax agreed with the Linköping University as CRML compiler developers.

Internally, the NUWARD™ project is a possible application of the work carried out during the master thesis. However, as EDF R&D is only involved in specific parts of the NUWARD™ project which is already at the operational (i.e. engineering) level, my work is indeed EDF's as a research action but is separate from the NUWARD™ project team. I did not only work on the NUWARD™ application case during the master thesis, but first on a simpler system (Intermediate Cooling System (ICS)) because it was an example already used in the EMBrACE project and well understood in terms of operation and requirements.

1.8 Outline

Chapter 2 explains the framework for requirements-centered system design and verification. This will be a theoretical part necessary to understand the graphical method defined in Chapter 3. The graphical method is then applied to the secondary circuit of a SMR in Chapter 4.

Chapter 2

Framework for the design and verification of complex systems

2.1 EDF's background on model-based system engineering

2.1.1 Motivation and start of requirements modelling at EDF

EDF has a long history of using system level modelling and simulation (*0D/1D*) to understand and demonstrate the behaviour of energy systems in different domains such as nuclear power plants, power grids or even building energy modelling. At the beginning of the twenty-first century, the Modelica language began to be used at EDF for this purpose [8].

Modelica expresses the behaviour of the system through the physical equations of the components, but it does not explicitly express the purpose for which the model is made. Indeed, the requirements concerning, for example the cavitation of a pump, were not mentioned in the Modelica model. A first attempt was to implement a dedicated constraint section in Modelica (ITEA EUROSYSLIB project (2007 – 2010)) but this solution encountered several problems. In particular, the requirements must have the notion of quantifiers ('No pump in the system must cavitate' or 'At least one in the system must be started') but this was not implemented in this section. Another problem was that Modelica expresses the physical state of the pump (its mass flow, pressure, etc.) but not its operational state (e.g. the pump is started or not). Thus, it is concluded that the behavioural and requirement models should be separated. Two representations of the pump will then coexist: one in the behavioural model and one in the requirements model. These two representations will be linked by 'bindings' (see Section 2.3.5).

In parallel, Airbus (a European multinational aerospace company) studied the possibility of linking SysML to Modelica [13] during the ITEA OPENPROD project (2009 - 2012). The requirements model would be made with Modelica blocks within SysML and the verification would be done in parallel with the simulation of the behaviour model in Modelica [5, 13]. It turned out that ModelicaML was difficult to use and did not solve the problems related to quantifiers.

The conclusion of the ITEA EUROSYSLIB project on the separation of the behavioural model and the requirements model led to the development of a new language for requirements modelling by the ITEA MODRIO project (2012-2016). Different notions had to be kept in mind for the development of this language:

- the syntax must be close to natural language
- the language must handle time periods and probabilistic aspects
- the language must handle quantifiers and be object oriented
- it should be possible to automatically generate test sequences from the constraints that represent assumptions on the system

2.1.2 Development of FORM-L, ReqSysPro and ETL

Thus, EDF developed in 2013 a new language called FORM-L with an industrial application in mind (in particular the Backup Power System after the Fukushima accident). It was proposed to the Modelica community, they refused to develop a new compiler, but as interested, offered to extend Modelica with the notion of "blocks as functions". EDF then developed a Modelica requirements library called ReqSysPro. Experimentation proved that a 4-valued Boolean logic (`true`, `false`, `undecided` and `undefined`) was required to rigorously evaluate requirements on (continuous) time periods and to be able to combine them. The semantics of the temporal aspects of FORM-L based on the 4-valued Boolean logic has been called Extended Temporal Language (ETL) [4]. A first FORM-L compiler developed by Inria and SciWorks Technologies on an EDF contract was made to generate ReqSysPro blocks from a FORM-L model. It was delivered in 2021 and demonstrates having a FORM-L to Modelica compiler. However, the compiler suffers from the drawback that it must be modified each time a new function is added to FORM-L, or if the name of a ReqSysPro block used by the compiler is modified. So it cannot be a long-lived stable compiler..

2.1.3 Emergence of CRML and current situation

To overcome this problem, a new language called CRML is being developed in the framework of the ITEA3 project EMBrACE (2019-2022). The idea is to add the notion of functions (called operators in CRML) to be able to build complex functions from a limited number of elementary native functions. CRML still uses the ETL semantics and its 4-valued Boolean logic [4] to rigorously evaluate requirements and implements FORM-L as a library of functions (called 'operators' in CRML) to express in a more readable way requirements (see Figure 2.1.1).

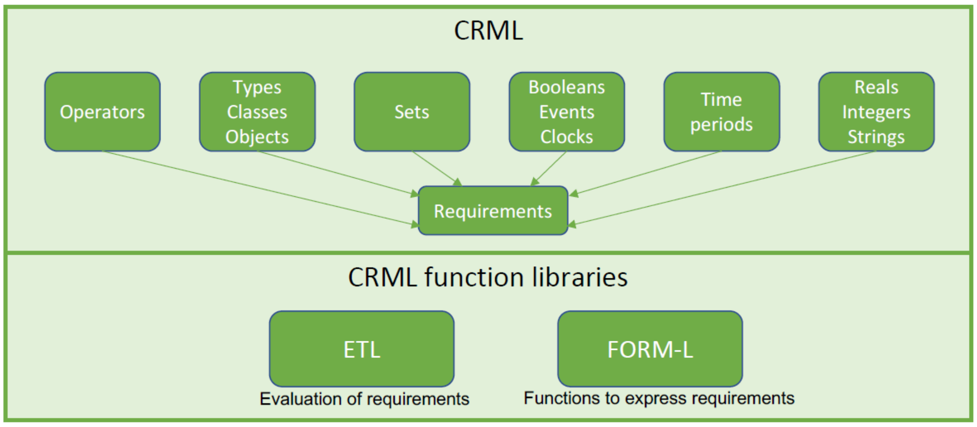


Figure 2.1.1: Architecture of CRML language

Version 1.1 of CRML specification has been released in Dec. 2022 and is publicly available here: https://www.embrace-project.org/specification/CRML.pdf.

Tool support is still under development: A Modelica library called `CRML.mo` has been developed by EDF. It is derived from ReqSysPro and validates ETL in CRML. A CRML to Modelica compiler is currently under development by Linköping University. A first prototype could be found here: https://github.com/lenaRB/crml-compiler. More details on CRML are given in Section 2.3.3.

The next major step was the development of a methodology to generate the requirements model graphically, which was the objective of this master thesis.

2.2 Requirements modelling

2.2.1 Requirements definition

The requirements definition process starts with stakeholder expectations, often expressed in natural language. Only those that can be translated into a set of technical requirements and therefore used in requirements models (see Section 2.3.3) will be considered here. Indeed, not all the requirements written in the specification will be considered because the CRML focuses mainly on dynamic requirements which therefore require a simulation to be verified. It is an iterative process to understand and define stakeholder expectations, project requirements, component and sub-component requirements.

Stakeholder expectations can be found in different documents such as interaction diagrams between the different modules of the system (see Figure 2.2.1). These are expressed in natural language and give only a few details about what is desired. Thus, they cannot be used as such in simulation. They represent high-level requirements and expectations.

Figure 2.2.1: Example of a translation of an interaction diagram for the interface between primary and secondary circuit for NUWARD™ (similar to what can be done with Capella software and in particular what they call dataflows)

These high-level requirements help to understand the technical problem and to determine the boundaries of the project. Boundaries can be defined for different reasons: choices that are already made and cannot be changed, interfaces with which the system must interact or overall expectations about the range of use and behaviour of the system within its environment. By understanding the expectations of the system and its design, it is possible to define a set of technical requirements and measures. Technical requirements are a description of the system and what needs to be validated in order to be approved by the stakeholders. Technical measures are measures that must be monitored to determine whether or not the defined technical requirements are verified. It represents the system variables of interest of the requirements model.

The set of technical requirements includes functional, performance and interface requirements. Functional requirements define the function of the system that is necessary to achieve the objectives. The performance requirements define quantitatively how the functional requirements must be achieved to meet the objectives. Consideration of performance requirements that are too restrictive prevents the investigation of potential design solutions. Performance requirements can be defined with a threshold value or with a desired baseline level of performance for the system. Interface requirements refer to the requirements that the system has between its different components. These can be different types of interfaces such as data, operational command and control, but the focus hereafter will be on physical requirements such as fluid, mechanical or thermal interfaces. For example, conservation of mass in a flow or conservation of energy will be discussed here. Physical requirements are less frequently mentioned in general in requirements verification.

Technical requirements are therefore very important, as they cover different types of requirements for the project, such as safety, functional or reliability requirements.

An example of a high level NUWARD™ requirement is *"F02: Convert steam to electricity". A technical requirement derived from this could be "In normal operation (DBC1), the plant shall produce at least 340 MWe"*." It is a functional requirement with its associated performance requirement.

2.2.2 Formalization of requirements

A requirement can be seen as four parts [5, 11]. It is a statement imposing a constraint (*WHAT*) on objects (*WHERE*) at particular times (*WHEN*) that must be verified with a certain precision (*HOW WELL*):

- Spatial Locators (*WHERE*): it defines which object or set of objects is concerned by the property. The term "spatial" does not refer to the geometric location of the object, but rather to the object selected according to different types of criteria.
- Time Locators (*WHEN*) : it defines the time period during which the property must be fulfilled. Time periods can be discrete or continuous (and may be overlapping). They are defined by opening and closing events.
- Conditions to be fulfilled (*WHAT*): It defines the constraint imposed on the object or set of objects during the time periods defined by the time locators. It can be a continuous condition that must be true during the whole time period or a discrete condition that must be true at the closing event.
- Probabilistic constraint (*HOW WELL*): it defines a probabilistic constraint of the property to be verified.

At a given instant, the evaluation of the requirement is `undefined` when it is not covered by its time locators. It is `undecided` when no definite conclusion can be made regarding the satisfaction of the property. Otherwise, it is either `true` or `false` (see Figure 2.2.2).

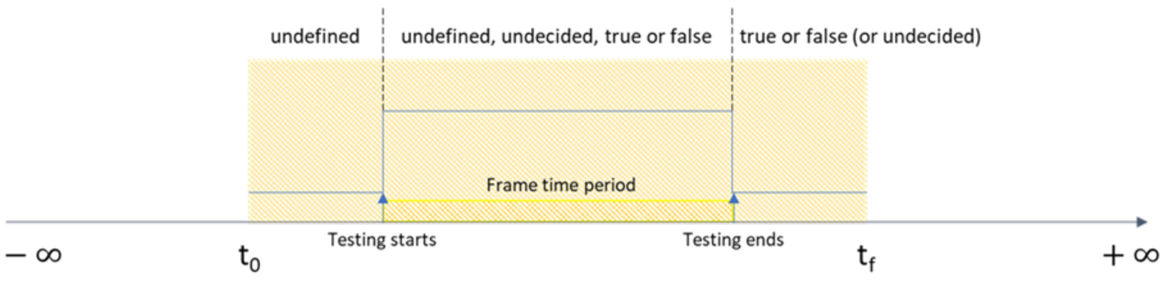
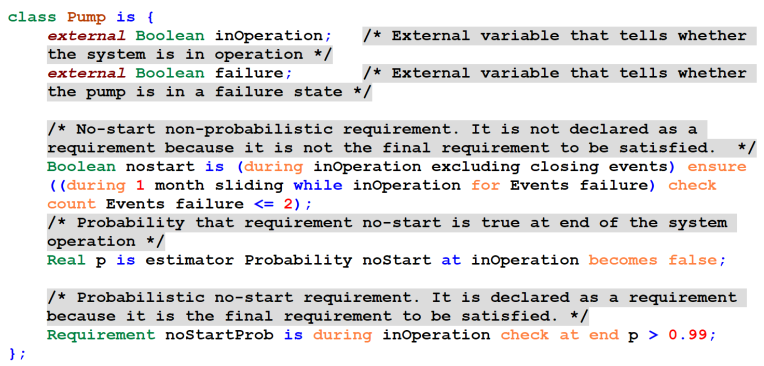


Figure 2.2.2: 4-Boolean states during a time period

An example of a property could be: "*Requirement* `noStartProb`*: While the system is in operation, there should not be more than two pump failures in a sliding time period of one month with a probability greater than 99%*". The translation into CRML is shown Figure 2.2.3. The class Pump is defined with the required external variables (Boolean `inOperation` and failure) and the requirement `noStartProb`.



```
class Pump is {
    external Boolean inOperation;    /* External variable that tells whether
    the system is in operation */
    external Boolean failure;        /* External variable that tells whether
    the pump is in a failure state */

    /* No-start non-probabilistic requirement. It is not declared as a
    requirement because it is not the final requirement to be satisfied.  */
    Boolean nostart is (during inOperation excluding closing events) ensure
    ((during 1 month sliding while inOperation for Events failure) check
    count Events failure <= 2);
    /* Probability that requirement no-start is true at end of the system
    operation */
    Real p is estimator Probability noStart at inOperation becomes false;

    /* Probabilistic no-start requirement. It is declared as a requirement
    because it is the final requirement to be satisfied. */
    Requirement noStartProb is during inOperation check at end p > 0.99;
};
```

Figure 2.2.3: Definition of a class in CRML with two external variables and a requirement

For the sake of simplicity, the requirements present in the following section will be simpler. Physical properties will be verified at interface to be in a certain interval or under/over a threshold for example. Functional requirements such as the non-cavitation of a pump will also be mentioned.

However, it should be kept in mind that more complex requirements such as `noStartProb` could have been used in the same way. In addition, requirements are also often expressed as a set of partial requirements to express the different situations in which the system may be as seen in Figure 2.2.3. An example of typical requirement to ensure is that:

- The system should stay within its normal operating domain.
- If partial requirement 1 above fails, then the system should go back to its normal operating domain within a given time delay.
- If partial requirement 2 above fails, or if partial requirement 1 fails with a too high failure rate, then the system should go to a safe backup state within a given time delay.
- The complete requirement made of the conjunction of partial requirements 1, 2 and 3 should be satisfied with a given probability (e.g., > 99.99%).

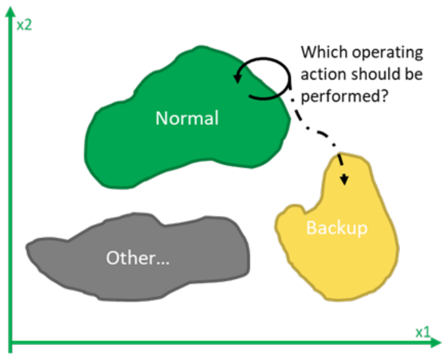


Figure 2.2.4: Different situations in which the system may be regarding requirements

2.3 Verification model: a way to use requirements in a design process

The purpose of the verification model is to verify the design of the system against the requirements. To do this, three different models are needed: the behavioural model, the architectural model and the requirements model [3]. The verification principle of the requirements model is illustrated in the figure 2.3.1.

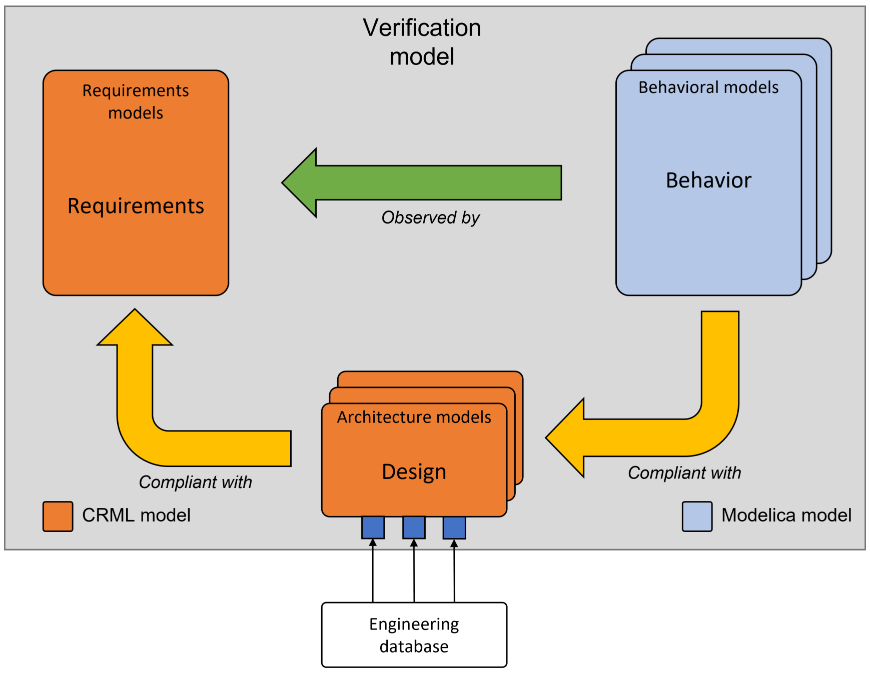


Figure 2.3.1: Verification Model

Depending on the different scenarios to be represented, several architectural models can be created for a given set of requirements. There are also several possible behavioural models to conform to a given design. Finally, the fundamental principle of verification model is the use of the requirements model as an observer of the behavioural model to detect violations in the requirements.

2.3.1 Behavioural models

The behavioural model is a mathematical model predicting the dynamics of the system based on a usage scenario of the system of interest and different parameters that are to be tested. The evolution of the dynamics is obtained from the simulation of the system. For the illustrative example of the graphical method, the behavioural model has been developed in Modelica with the EDF library ThermoSysPro.

Modelica is a multi-domain modelling language for complex systems. ThermoSysPro is a Modelica library developed at EDF for thermal-hydraulic modelling.

The model is built by assembling components belonging to ThermoSysPro [8] and connecting them according to certain rules to express the relationships between them. This allows the model to be well formulated mathematically and physically. An example of a behavioural model is shown in Figure 2.3.2, which is a model of an ICS for equipment in a Pressurised Water Reactor (PWR) nuclear power plant. ThermoSysPro components icons are close to the Piping and Instrumentation Diagram (P&ID) standards as shown in Figure 2.3.2. Pumps are shown here inside a red circle, ControlValves inside a green circle and HeatExchangers inside a blue rectangle.

2.3.2 Architectural models

In order to verify system requirements, sufficient information on the system architecture must be provided. These are static properties of the system which can be found in engineering databases. For example, if a pump is used in the system, this model can provide manufacturer data on the pump such as the required Net Positive Suction Head (NPSH).

Several architectural models can be used in case different technologies or architectures want to be tested.

2.3.3 Requirements model

The requirement model is the model that is intended to be verified against the others. It is a formal representation of the system requirements. It expresses the requirements that need to be verified, taking into account the assumptions made on the system's environment and its interfaces.

Taking the example of the ICS, consider a global technical system requirement specifying that "*the ICS shall remove a maximum heating power of 25 MW*". In this case the system's internal

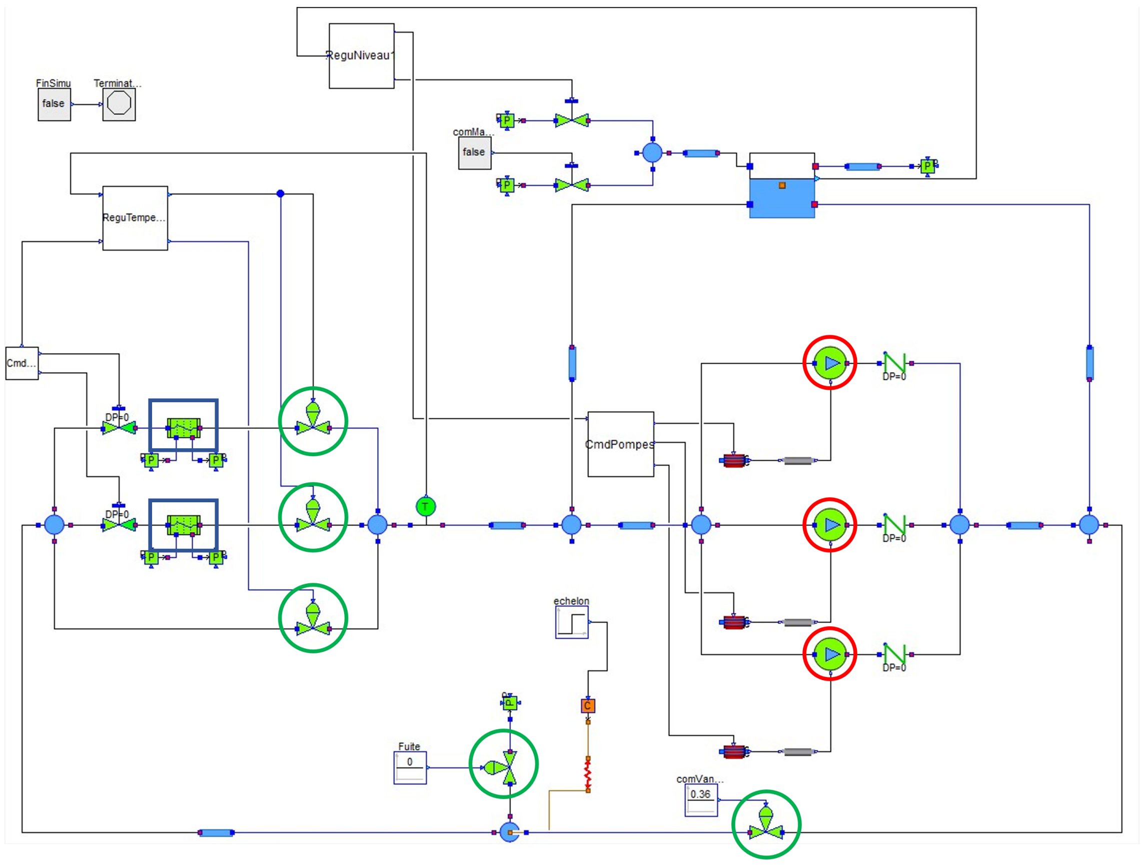


Figure 2.3.2: Example of a behavioral model of a cooling system built with ThermoSysPro

structure is not needed. Only a few external variables (the ”technical measure” of evacuated heat, see Section 2.2) are needed to represent the heat flow interface that the system has.

However, other requirements require the identification of certain components of the system. Consider for example the following technical requirement: *“The temperature at the outlet of the heat exchangers shall be close to 17°C within a ±1°C margin with a confidence rate of x%.”* This requirement is an objective to be achieved to operate the system at best. This requires knowledge of the components: the system is no longer a black box with respect to the requirement model (one should know that the system is composed of at least one heat exchanger). The requirement model is written in CRML. CRML is an object-oriented language developed at EDF to formally express dynamic and realistic requirements applied to multi-physics components in a behavioural model of Cyber-Physical Systems (CPS)s. Object-oriented programming is a programming model that organises its design around the notion of classes and objects. It is a way of structuring software into small pieces of reusable code (classes) that are then used to create instances of objects (instantiation). Like most object-oriented languages, it has the notion of inheritance. Inheritance is the ability to create a new class from an existing class that will inherit its various attributes so that the new class can then be refined.

It also has the more common notion of libraries which will be used in the method.

As the language is still under development, the syntax may have (slightly) changed from the specification document provided [2]. Some examples of CRML code can be found in this report, others can be found in the Appendix B.

As said in the introduction 1.3, the aim of the master thesis is to create a graphical method for the representation of requirements and the evaluation of different architectures. It is thus a question of making a graphic representation of the requirements model. It will be seen how the method allows a link between requirements and architectures in Chapter 3.

2.3.4 Process for model development

In practice, the three types of models described above (requirements, architectural and behavioural) are often developed independently by different teams.

Indeed, the behavioural model is made by different expert engineers (each with their own fields of expertise in thermal-hydraulics, neutronics, control, etc.) with their own professional tools (0D/1D tools such as Modelica, Matlab/Simulink, FlowMaster, ...or 2D/3D tools such as Fluent...). In addition, each engineer will make assumptions about the interface of his part of the system. For example, the neutronics expert will assume that the water entering the PWR core has a temperature in a certain range and will guarantee to the steam generator engineer that the water leaving the core has a temperature in another range [1].

The architectural model must be created with mechanical and Instrumentation and Control (I&C) expertise. It should capture the different possible designs of the system. It will be crucial to have different conception choices in order to find the best possible architecture for the system.

The requirements model is elaborated by the experts in systems operation. Its purpose is to capture the system's requirements. Those may be modified when operating constraints evolve such as safety or environmental regulations. The requirements will be given to him by different parties: the customer with the associated specifications, the engineers making the subsystems...

2.3.5 Bindings: a way to connect models

The three types of models (requirements, architectural, behavioural) must understand each other in order to carry out the verification of the requirements. However, they are very different in nature. On the one hand, the requirements and architectural models provide a functional description of the system and are not linked to specific components of the simulation. Indeed, a pump defined in the requirements model can be described in several ways in the behavioural model because different components may or may not be part of it in this model as shown in Figure 2.3.3.

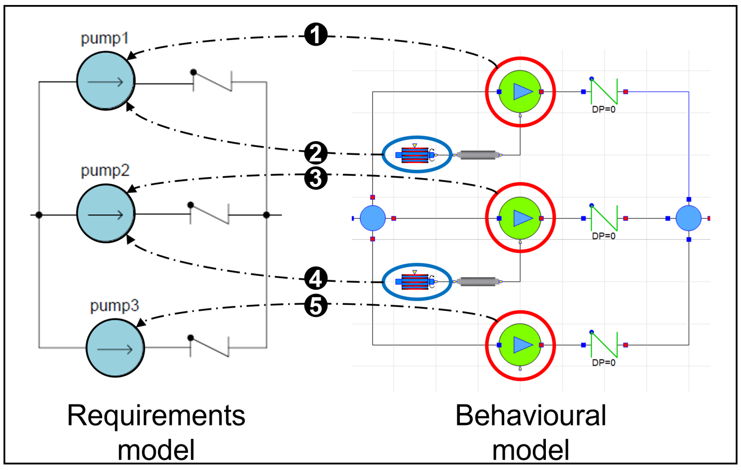


Figure 2.3.3: Instance binding between the requirements model (left) and the behavioural model (right) for the pumps

On the other hand, the behaviour is described by physical equations and control laws for each instance of the components, and all these instances must verify requirements. Moreover, these different models must be linked together without having to modify them as behavioural models are expensive and were developed primarily for design purposes. This is made possible by observation operators and bindings (Figure 2.3.4). Bindings translate the physical concepts of the behavioural model into the functional concepts of the requirements model. For example, the functional notion of starting a pump can be defined by different physical observation operators:

- If the motor of the pump is started;
- If a positive torque is applied to the pump;
- If a sufficient pressure between the inlet and outlet of the pump is observed.

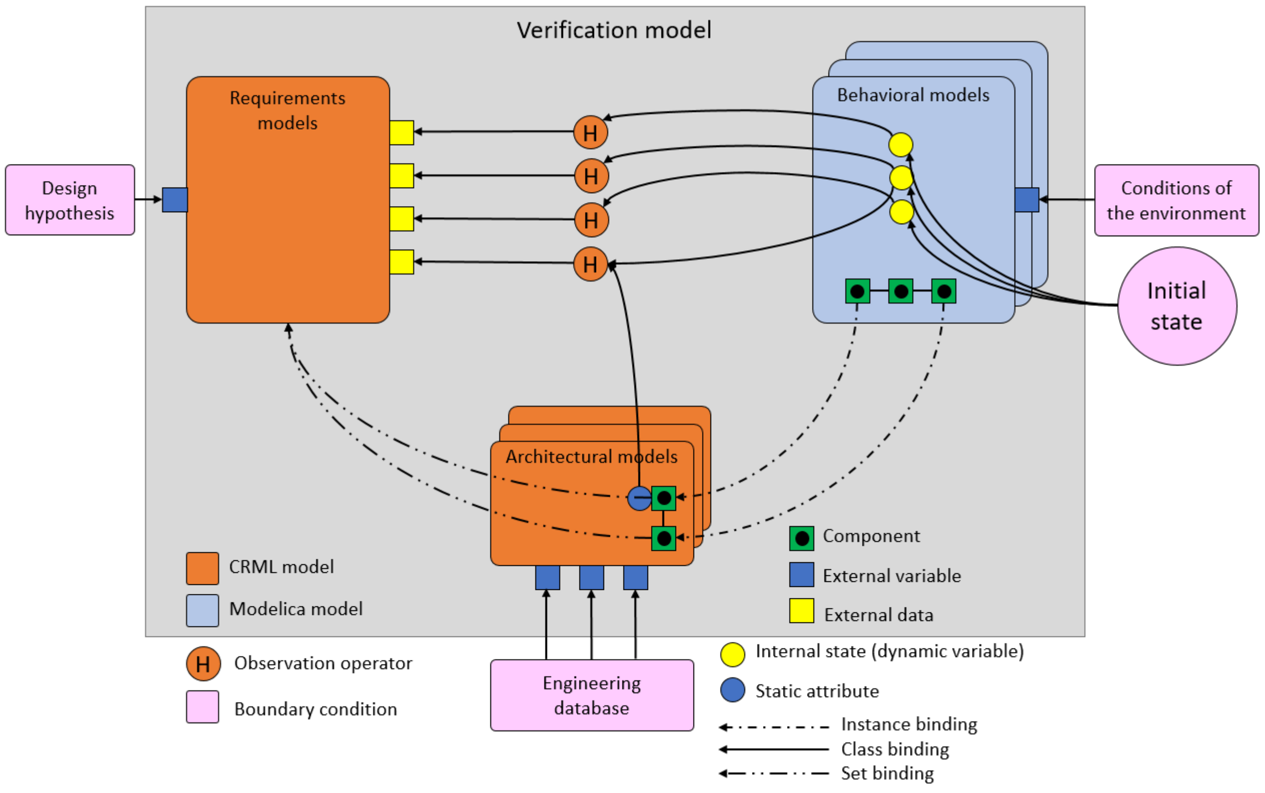


Figure 2.3.4: Complete configuration of the verification model with bindings and observation operators [3]

Chapter 3

Towards a new graphical method for systems design

In this chapter, a detailed description of the graphical method conceived during the internship is given. First, the objectives of the method are given in order to understand the task. Then the method will be presented in a general use case. An example will be given in the next chapter.

3.1 Goal of the method

This method aims at graphically represent any multiphysics energy system and its associated requirements to serve as a guiding framework for system design. The application example of the internship was that of the secondary circuit of nuclear power plant for the new project of SMR of EDF NUWARD™. However, it should be kept in mind that any other energy systems mixing different physical domains could a priori be also represented by this method.

In organising requirements in such models, the aim was to be able to understand where a requirement comes from. This may be a constraint on the operation of the component itself (for example, a pump must not cavitate), a constraint on the choice of a particular component (a manufacturer's constraint), the expectations of the specification, etc. By understanding this, one can also understand the margins taken in the project and therefore reduce them. Indeed margins, when they are accumulated, can lead to excessive costs compared to the standards that need to be met. Thus, having a better knowledge on those could lead to finer modelling of the system to verify the requirements but even to changes in the design. Furthermore, we would like to be able to represent different possible architectures of the system (different configurations that are to be tested). Indeed, the project NUWARD™ emphasises the modularity of the project in the case of different uses. Thus, this method should follow the same indications allowing a modular representation. The ambition of this project is not to use this method to make only simple parametric studies of the system but to consider different scenarios of use of the system (see section 4.1). In addition, the different architectures envisaged must be traceable. Indeed, the choices made in the design must be justifiable and thus an architecture comes from the modification of other previous architectures. This monitoring of modifications is essential in the storage of knowledge.

The method helps the engineer in his design with regard to requirements verification. In order to facilitate the adoption of the requirements models by the engineer and not only by the CRML code, the method proposes a structuring of the CRML code through the use of graphical (and hence hopefully intuitive) means to ease its appropriation by the engineer. Indeed, as the requirements would be written in CRML, we would like the engineer dealing with the requirements not to have to manipulate the code, that the code is just an intermediary to go towards the verification, hence the development of a graphical method. It is part of a wider method of requirements verification (see Section 2.3 and Figure 3.1.1) and therefore allow for the automation of the creation of a CRML template for the requirement model to carry out the verification (see Figure 2.3.1). Other SE tools such as SysML/UML are only descriptive and do not allow the creation of computable code. The automation of the transformation of the graphical method into CRML is not part of the internship. This was done manually for the examples studied in the report. However, even once such graphics to CRML translation will be automated, the method should not be considered as an automatic translation of natural language into CRML since natural language is rarely unambiguous and human is often needed to better translate it in a more formal way). The aim is also to provide a library of example components whose requirements are already implemented.

Finally, in order for it to be better understood and accepted by the scientific community, and in particular the Systems Engineering (SE) community, it has been made as close as possible to the Unified Modelling Language Unified Modelling Language (UML) 2.5 and SysML 1.6 syntaxes. These are graphical syntaxes that already exist and are widely used in various engineering fields.

3.2 The new graphical method for expressing requirements

This method illustrates how a requirements model can be made graphically to represent the different notions mentioned in section 2.2. As there is no automation of a CRML template yet, it can be drawn on any software or graphic tool. Furthermore, the changes to the CRML syntax are not relevant to the method as the method itself does not change. It is not made

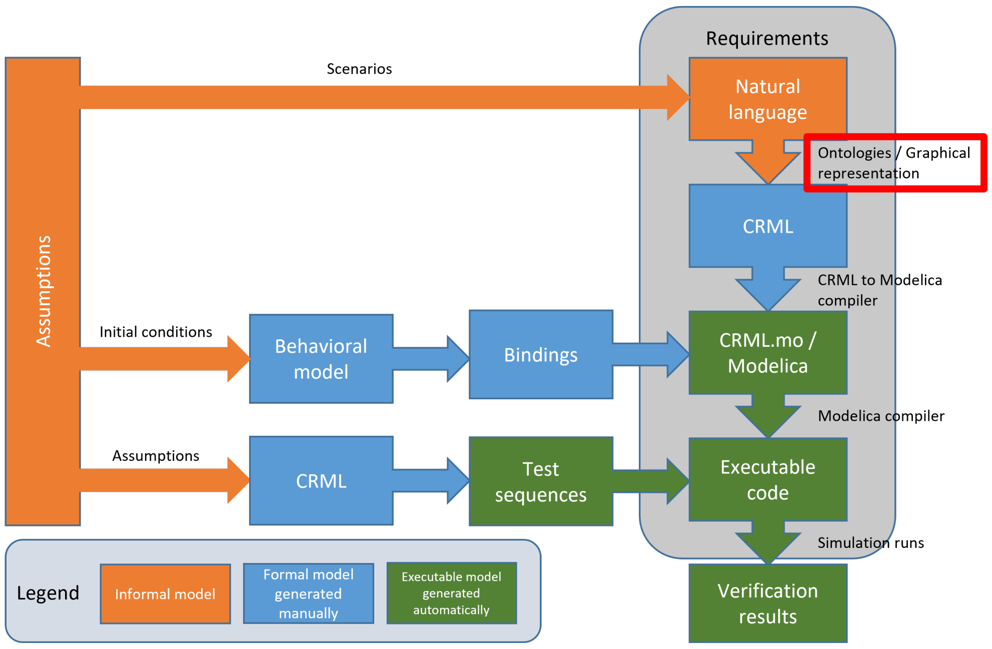


Figure 3.1.1: Verification process. The graphical method discussed in this chapter is in red.

for CRML especially, it is made around CRML (the graphical method aims more at structuring requirements and associated input data each time they are refined during the project lifecycle than at expressing requirements in a certain manner).

3.2.1 Main principles

The method consists of three different diagrams: the inheritance diagram, the internal structure diagram and the instance diagrams, each with its own purpose for creating the final requirements model in CRML.

The inheritance and internal structure diagrams are used to create a CRML library. It groups the classes of the various components modelled without instantiating objects. The notion of class is briefly explained in the section 2.3.3. In the inheritance diagram, the attributes and requirements are defined for each class. The internal structure diagram represents each component present in the inheritance diagram but specifies which sub-components it is made of. Thus, the two diagrams represent the same classes but have a different point of view: the inheritance diagram is used to describe the attributes of a class, while the internal structure diagram is used to describe a class and how it is designed.

Instance diagrams represent the architecture and the components that have been chosen for testing. It is in this diagram that they are instantiated. Different instance diagrams can coexist with the same inheritance and internal structure diagram if one wishes to test different architectures.

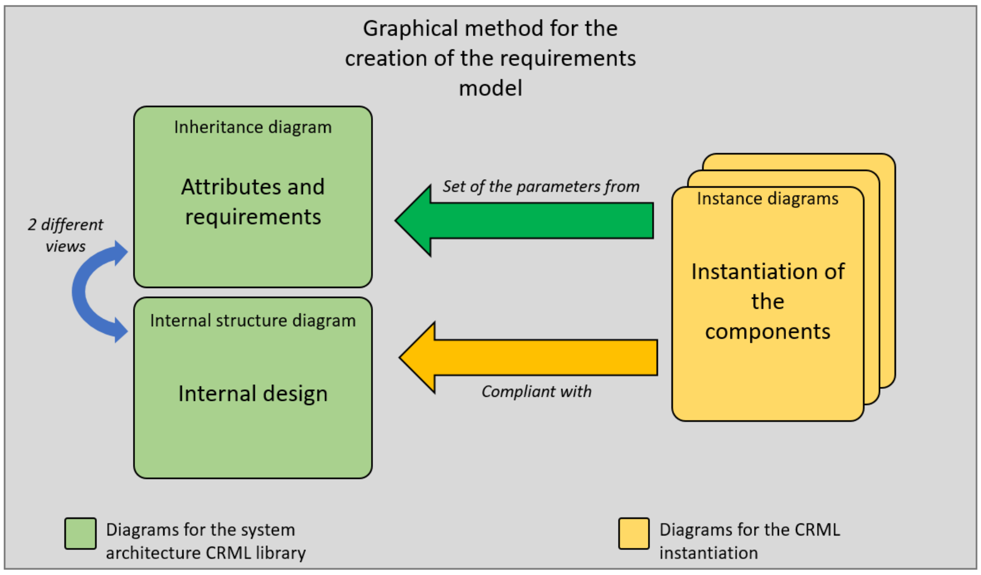

Graphical method for the creation of the requirements model
Inheritance diagram
Attributes and requirements
Internal structure diagram
Internal design
2 different views
Set of the parameters from
Compliant with
Instance diagrams
Instantiation of the components
Diagrams for the system architecture CRML library
Diagrams for the CRML instantiation

Figure 3.2.1: Diagrams used in the graphical method

The three diagrams will be presented in the following subsections. As they are part of a graphical method, the use of an example will be useful to best describe them. As an example, consider an ICS in a nuclear power plant. The corresponding behavior model is shown Figure 2.3.2.

The purpose of the system is to cool plant equipments called ServedSystem (Figure 3.2.2). To do this, the system is composed of heat exchangers interacting with a cold source and various pumps for the fluid (water) to circulate in the circuit. The pumps are considered in a PumpBlock component in order to test different architectures (how many pumps? in series or in parallel?). This component will be used as an example to illustrate the diagrams. Its mission is fluid circulation in the system.

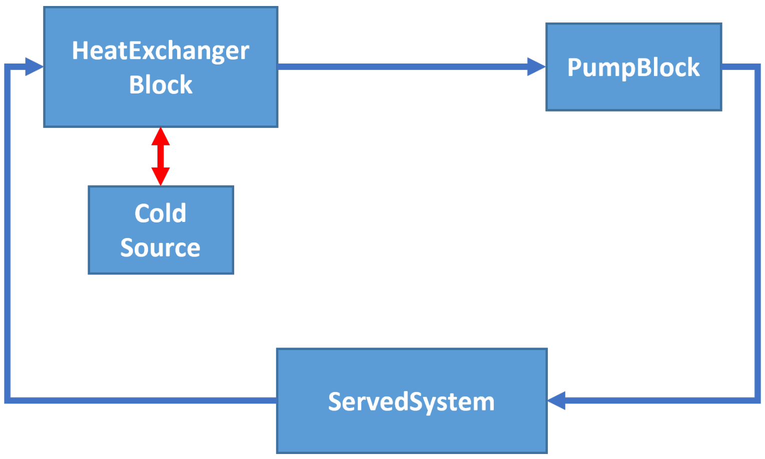


Figure 3.2.2: Components of a simplified ICS

3.2.2 Inheritance diagram

The inheritance diagram is used to declare all the attributes and requirements of the classes of components or sub-systems considered in the system. In the example above, classes will be created for the ServedSystem, the ColdSource, the PumpBlock, the Pumps (since it is a sub-component of the PumpBlock), the HeatExchangerBlock and the HeatExchangers (since it is a sub-component of the HeatExchangerBlock). The choice could be made later to refine the model and to model, for example, the different valves of the system. In this case, new classes would be created.

The attributes to be defined in this diagram are all those needed in the requirements. At the beginning of the diagram creation process, the inheritance diagram should represent the different classes mentioned above and look like the Figure 3.2.3.

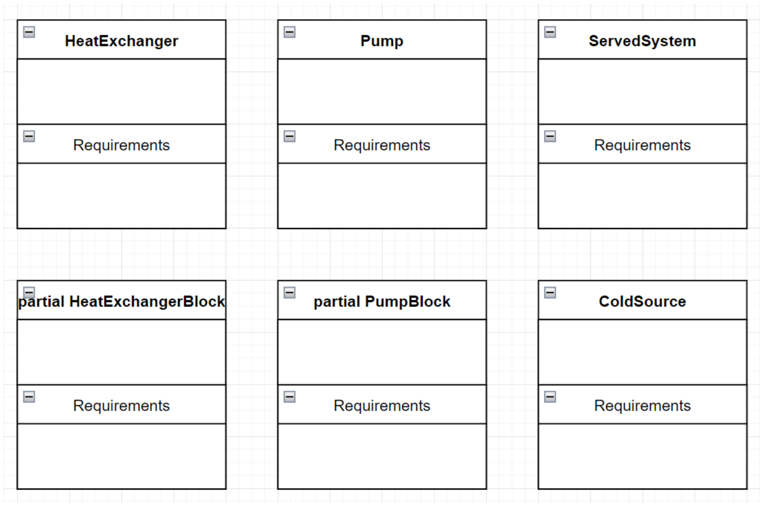


Figure 3.2.3: Inheritance diagram at the beginning of the diagram creation process

In CRML, classes can be partial. This means that they are not complete and cannot be instantiated as such. The choice of whether it is partial (and therefore whether it is complete or not) is left to the CRML engineer. A pump will be considered as an elementary component at this stage because no architecture of the interior of the pump needs to be defined. The class can therefore be instantiated as such. For the PumpBlock, it will be considered as a partial class because the internal architecture (how the pumps are arranged) must be defined. This will be explained later.

As CRML is an object-oriented language with the notion of inheritance, it will be used to simplify the declaration of classes. This means that if different pieces of equipment have the same attributes in common, they can inherit from the same class. For example, if both Pump,

HeatExchanger and ServedSystem have an attribute called ”`inOperation`” (indicating whether the equipment is in its operational state or not), both classes could inherit from an Equipment partial class that has the ”`inOperation`” attribute. The value of the operational state of the equipment would be defined by observation operators as mentioned in 2.3.5. Attribute “`inOperation`” is then defined as an external variable in the partial class Equipment. Graphically, the inheritance is written with an arrow marked ”Extends” (because the definition of Pump extends the one of the Equipment class, see Figure 3.2.4).

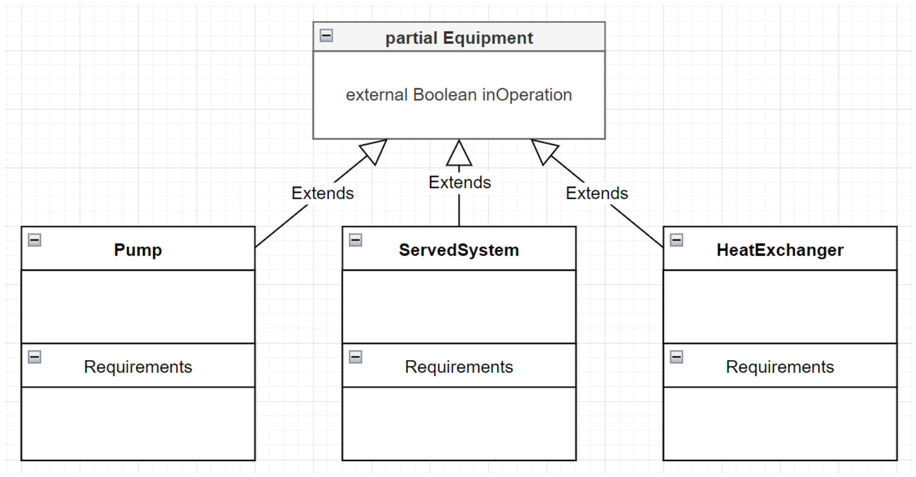


Figure 3.2.4: Graphical example of the notion of inheritance in the diagram

The previous figure shows the creation of a partial class with an external attribute, as well as three classes inheriting from this first one. The aim of the graphical method is to facilitate the appropriation of CRML by the engineer. Thus in the mid-term when supporting tools will be developed, Figure 3.2.4 would have its translation into CRML done automatically since the diagrams are meant to be formal. It would provide the CRML code shown in Figure 3.2.5.

```
partial class Equipment is {
    external Boolean inOperation;    // State in operation or not of the equipment
};

class Pump is {
} extends Equipment;

class ServedSystem is {
} extends Equipment;

class HeatExchanger is {
} extends Equipment;
```

Figure 3.2.5: Graphical example of the notion of inheritance in the diagram

The Pump, ServedSystem and HeatExchanger classes are empty in Figure 3.2.5 because no attributes are yet defined in Figure 3.2.4.

What remains to be done in the inheritance diagram is to complete all the classes with the requirements that we want to verify, and therefore to complete them with the associated variables required to evaluate such requirements. These requirements can be found in the specifications if they are global requirements on the system, by the engineers if they are specific characteristics of the system... Let's go back to the ICS example and consider a set of some simple requirements (see Figure 3.2.6).

- Req. 1: "*The ICS shall remove a maximum heating power of 25 MW*"
- Req. 2: "*In normal operation, the pumps in the system shall not cavitate*"
- Req. 3: "*In normal operation, the temperature at the outlet of the heat exchangers shall be close to 17°C within a ±1.5°C margin with a confidence rate of x%.*"
- Req. 4: "*In normal operation, the output mass flow rate of the pumps shall be less than 1000 kg/s*"

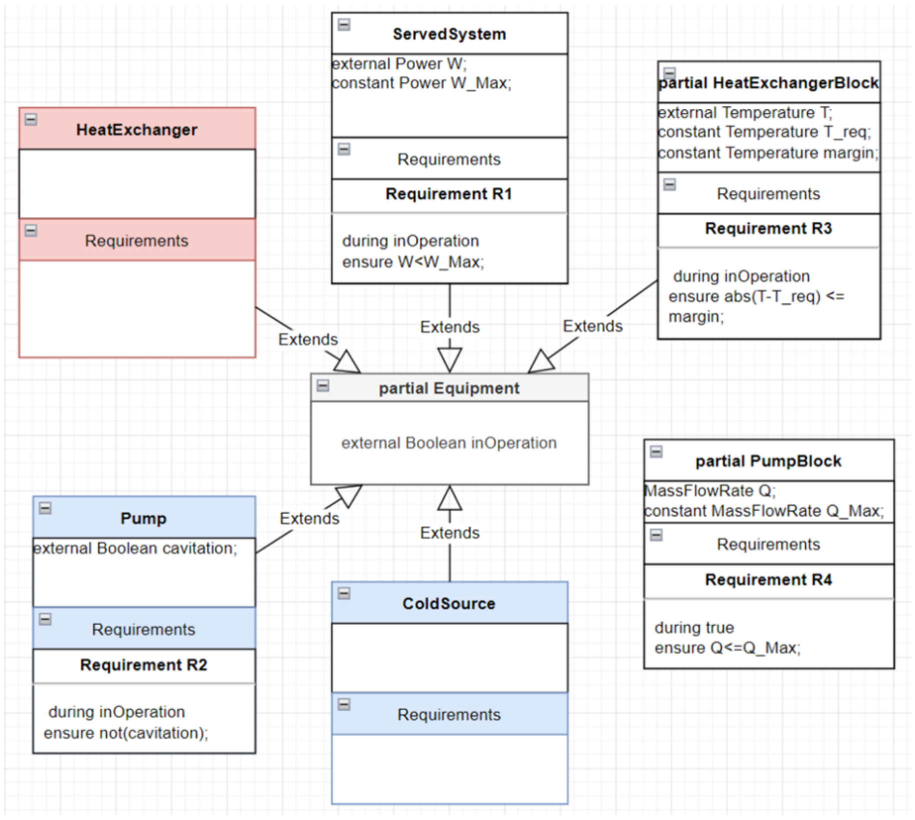


Figure 3.2.6: Requirements and attributes for the example of the ICS

The CRML code corresponding to the Equipment partial class and its extended classes of Figure 3.2.6 is shown in Figure 3.2.7.

Requirements from the specification may not be precise enough (See section 2.2) for the requirements model. Indeed, the desired accuracy is often unclear. For example, a requirement for this system might be Req. 5: *"The mass flow rate in the circuit shall be quasi-constant"*. It is necessary to ask how "quasi-constant" is defined and thus have clear technical requirements.

Finally, the partial classes requiring an internal structure must be completed in order to finish

```
partial class Equipment is {
    external Boolean inOperation;    // State in operation or not of the equipment
};

class Pump is {
    external Boolean cavitation;

    Requirement R2 is during inOperation ensure not(cavitation);
} extends Equipment;

class ServedSystem is {
    external Power W;
    constant Power W_max;

    Requirement R1 is during inOperation ensure W<W_Max;
} extends Equipment;

class HeatExchanger is {} extends Equipment;

class ColdSource is {} extends Equipment;

partial class PumpBlock is {
    MassFlowRate Q;
    constant MassFlowRate Q_Max;

    Requirement R4 is during true ensure Q<=Q_Max;
};
```

Figure 3.2.7: CRML example for the requirements and attributes for the ICS

the inheritance diagram. This phase is carried out in parallel with the process of creating the Internal Structure diagram because the two diagrams represent the same classes but from a different point of view. This is the case for the HeatExchangerBlock and PumpBlock partial classes. For the sake of simplicity, only one architecture will be considered for the HeatExchangerBlock (consisting of two HeatExchangers in parallel). Two architectures will be considered for the PumpBlock, one with two pumps in parallel and one with three pumps in parallel (see Figure 3.2.8).

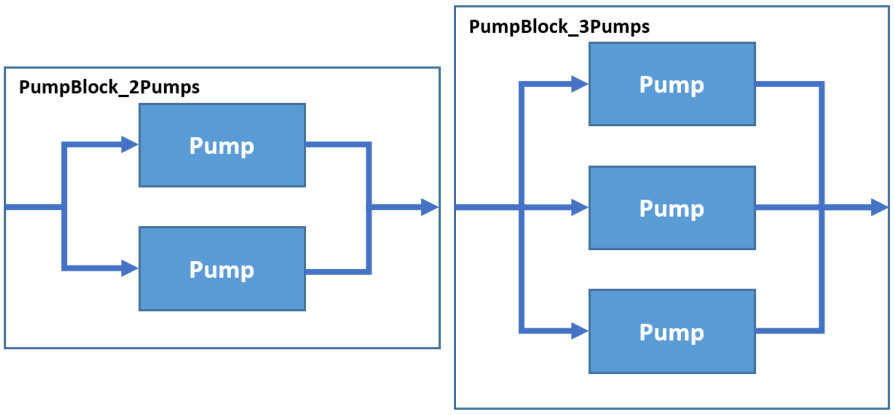


Figure 3.2.8: Figure showing the two different architectures considered for the example. This figure is not part of the inheritance diagram, it is for explanatory purposes only.

Two new classes are created, extending the partial class PumpBlock, so that the two new classes inherit the requirements and attributes. In these two classes, the internal classes used in the architecture under consideration are written (see Figure 3.2.9).

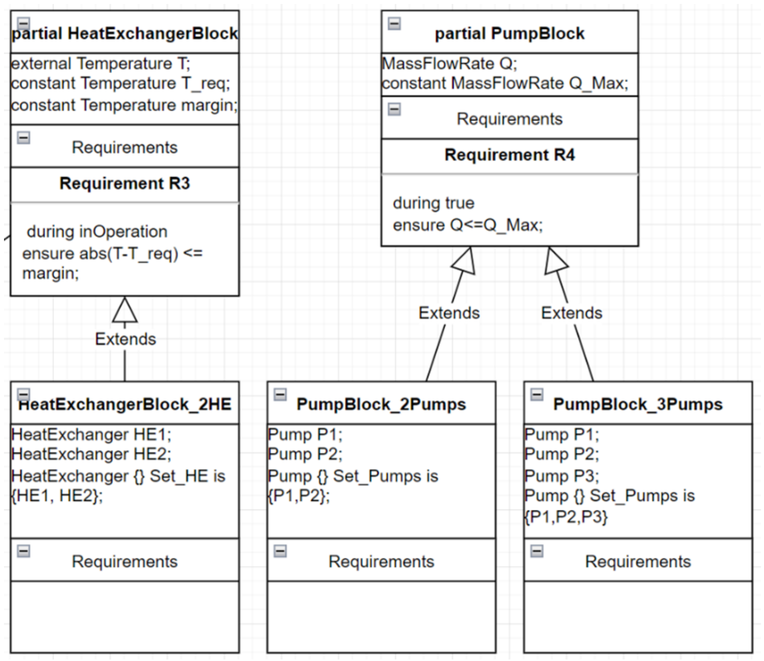


Figure 3.2.9: Inheritance diagram considering two different architectures for the partial class PumpBlock, and one for the partial class HeatExchangerBlock

The class with an internal structure is called the parent class. The classes contained within a parent class are called child classes. In this example, Pump is a child class of the parent classes PumpBlock_2Pumps and PumpBlock_3Pumps. The translation into CRML of PumpBlock and its child classes is shown in Figure 3.2.10. Partial classes are created with the required variables (external and constant attributes). The requirements are also defined in CRML. The components used in the architectures are also written (assuming that the corresponding classes have been created beforehand).

```
partial class PumpBlock is {
    MassFlowRate Q;
    constant MassFlowRate Q_Max;

    Requirement R4 is during true ensure Q<=Q_Max;
};

class PumpBlock_2Pumps is {
    Pump P1;
    Pump P2;
    Pump {} Set_Pumps is {P1,P2};
} extends PumpBlock;

class PumpBlock_3Pumps is {
    Pump P1;
    Pump P2;
    Pump P3;
    Pump {} Set_Pumps is {P1,P2,P3};
} extends PumpBlock;
```

Figure 3.2.10: CRML Translation of PumpBlock and its child classes from 3.2.9

Finally, the inheritance diagram for the example considered is shown in Figure 3.2.11 and in Appendix A.

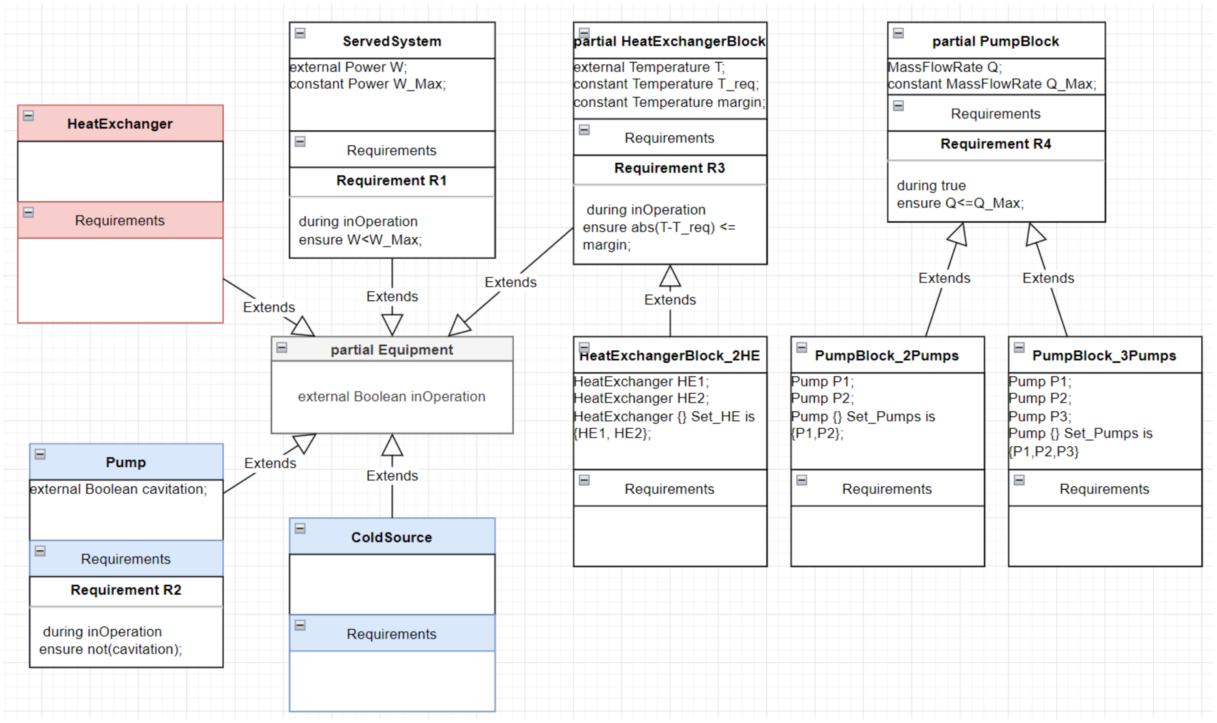


Figure 3.2.11: Inheritance diagram for the considered example of the ICS

3.2.3 Internal structure diagram

The internal structure diagram is used to define the internal structures of classes defined in the inheritance diagram. These two diagrams must be done in parallel. Indeed, when a partial class needs an internal structure to be an instantiable class, the internal structure will be defined in the corresponding diagram.

The first thing to do in the diagram is to represent the outermost view of the system considered. This is a representation of the figure 3.2.2. This will not be used in the automation of the CRML template but it is an aid for the reader to better understand the architecture. For the ICS example under consideration, the top view is shown in Figure 3.2.12.

Each class in the inheritance diagram is then described. If it is an elementary component, then it will have no internal structure but will still be present so that it is known to be an elementary component. If the parent class has an internal structure, it will be described with blocks of child classes inside the parent class (see Figure 3.2.13). These blocks of child classes are neither instances of classes, nor they are whole classes. They are called “individual parts” [6]. This term is inspired by the syntax UML for the composite structure diagram. If a parallel is drawn between this graphical method and the UML syntax, a parent class is a structured classifier. The internal structure diagram is the UML composite structure diagram. The instance diagram

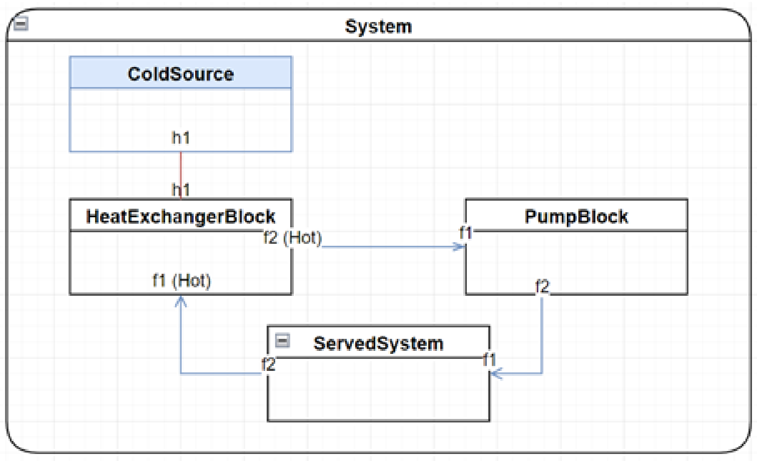


Figure 3.2.12: System top view in the Internal Structure Diagram

discussed in the next subsection is then an InstanceSpecification of the system described by the StructuredClassifier (see [12]). The PumpBlock example will be used again as example. Two architectures have been defined, one with two pumps in parallel, one with three pumps in parallel.

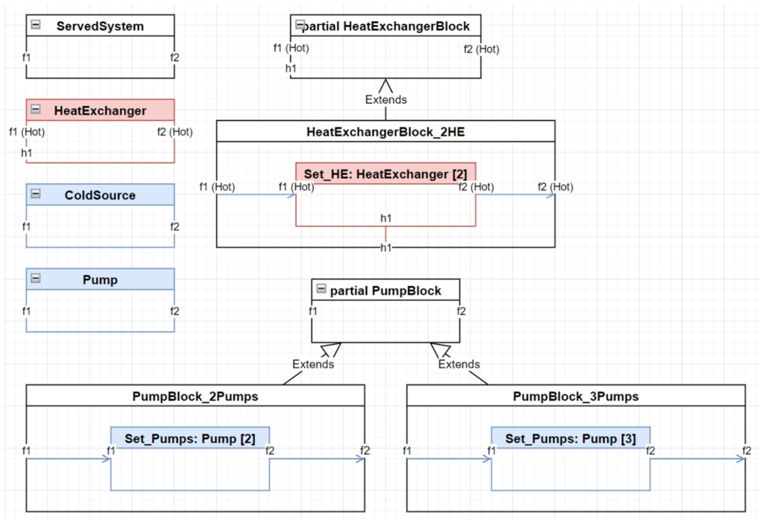


Figure 3.2.13: Internal structures of the different components in the ICS

If the components are in parallel, the multiplicity is indicated in brackets (see Set_Pumps and Set_HE in Figure 3.2.13). This is done to simplify the diagram. It is still possible to do this by placing three pumps in parallel as in Figure 3.2.14. Only the names of the different parts change, but this is specified in the inheritance diagram anyway.

The arrows on the internal structure diagram represent connectors between the components. This will not be used in the automation of the CRML template but it is an aid for the reader to better understand the architecture. Those connectors are connected to ports defined on each class. Ports are defined by a letter and a number:

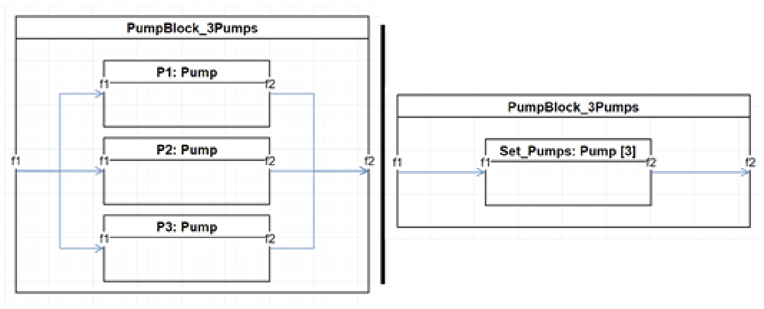


Figure 3.2.14: Two identical representations for parallel components

- If the connector represents an exchange between pieces of equipment, the number is even when it represents an input, odd when it represents an output. If the exchange is in both directions, two arrows are drawn
- Otherwise, the numbers have no role other than to differentiate them

In the Figure 3.2.12, most ports are fluid ports. The heat exchanger has for example one inlet for the hot side and one outlet for the hot side. This notion of port is also part of the UML syntax for StructuredClassifiers. The difference is that UML ports are required and provided interfaces. However, talking about RequiredInterfaces and ProvidedInterfaces for the interface of a physical system was not clear (interface of a pump for instance is not obvious/can change when the pump can be used in pump mode or in turbine mode for a TurbinePump). The notation used in UML (sockets and lollipops notation, Figure 3.2.15) has been tested but made the diagram unclear.

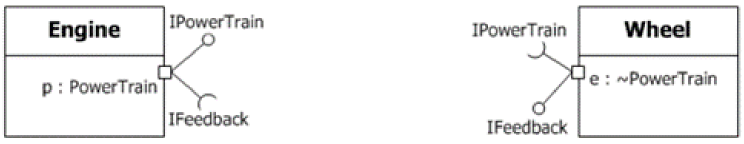


Figure 3.2.15: Sockets and Lollipops notation from UML [[12]], Engine has a RequiredInterface IFeedback and ProvidedInterface IPowerTrain, Wheel has a RequiredInterface IPowerTrain and ProvidedInterface IFeedback

Therefore, a different notation from UML with simple arrows (or lines) is finally used in the graphical method. The notion of RequiredInterfaces and ProvidedInterfaces could be interesting for the future implementation of the contract theory of [1]. The notion of ports used here is closer to the one used in SysML as it is also made to represent physical systems.

The VHDL language [9] (inspired by ADA [14]) was briefly discussed during the internship to get ideas for this part of the method. VHDL is a hardware description language for representing the behaviour and architecture of a digital electronic system, but does not have a graphical representation of the components (only a descriptive textual language). Several architectures can be written for the same component and thus different levels of abstraction are represented. Its notion of different types of ports (Boolean, current, etc.) is somewhat related to what is considered for ports in the new graphical method. VHDL ports are however defined as inputs and outputs only. This can be problematic as some components, such as the pump-turbine (reversible pumps used in pumped storage technology [15]) have reversible inputs and outputs. The graphical method therefore tends to be more general and not limited to inputs and outputs.

The complete internal structure diagram for the ICS example is presented in Figure 3.2.16.

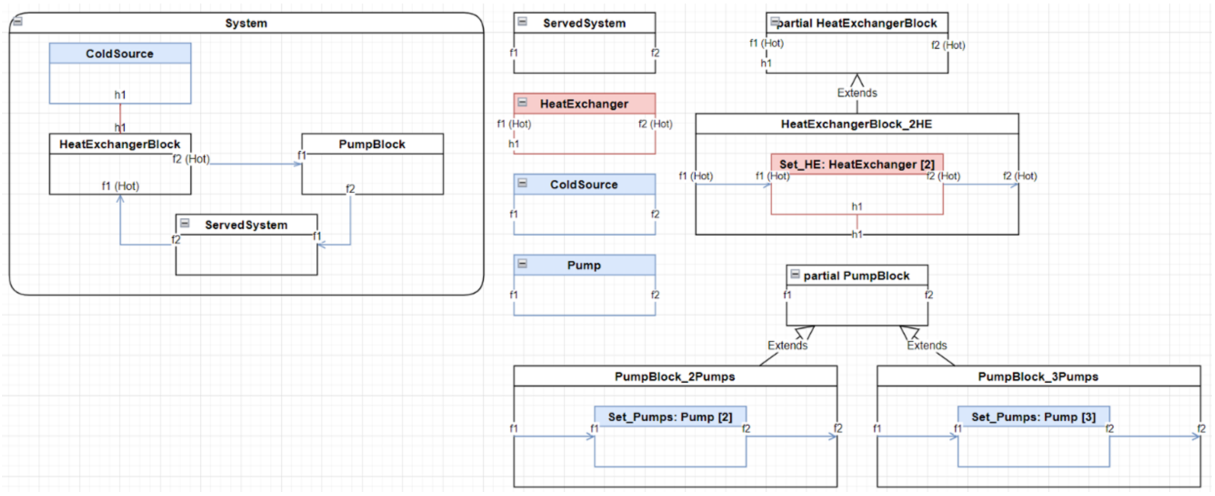


Figure 3.2.16: Internal structure diagram of the ICS

Redeclaration and refinement of the architecture

The strength of the method is that it can be easily adapted to a new architecture or to a finer level of precision when the design project progresses. Let's come back to the ICS example, for now, the pump is an elementary component, but the pump model could be refined by having different technologies and even different internal architectures. In this case, new classes will be created, inheriting the now partial Pump class.

Consider a new class CentrifugalPump, which will inherit the Pump class but may have more requirements (due to the technical characteristics of the new equipment considered). This is a refined definition of the Pump class. In this example, both classes are still elementary components. Equipment that contains a Pump instance may then need to be re-declared to have the new definition of the CentrigugalPump if required. To do this, CRML has the notion of re-declaration. This is done to redeclare an attribute (from Pump to CentrifugalPump for example) but can also change its name [2]. This can be done with the graphical method. It will be visible on both the Inheritance and InternalStructure diagrams. First of all, the new

Centrifugal Pump class is defined as an inheritance of the Pump class (Figure 3.2.17).

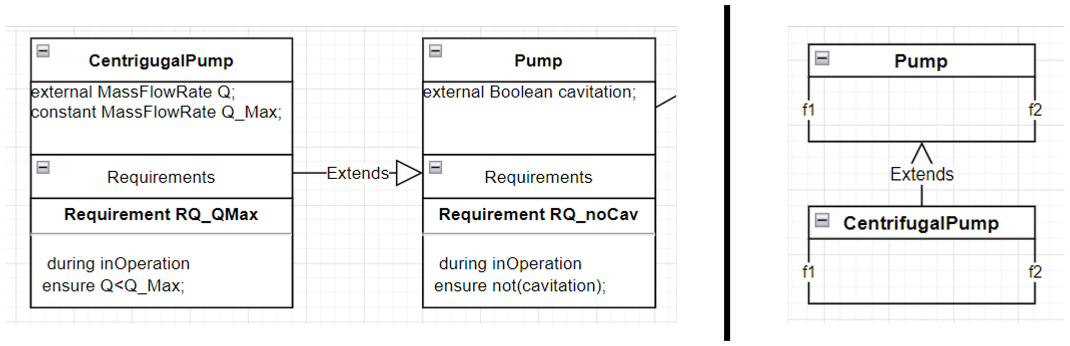
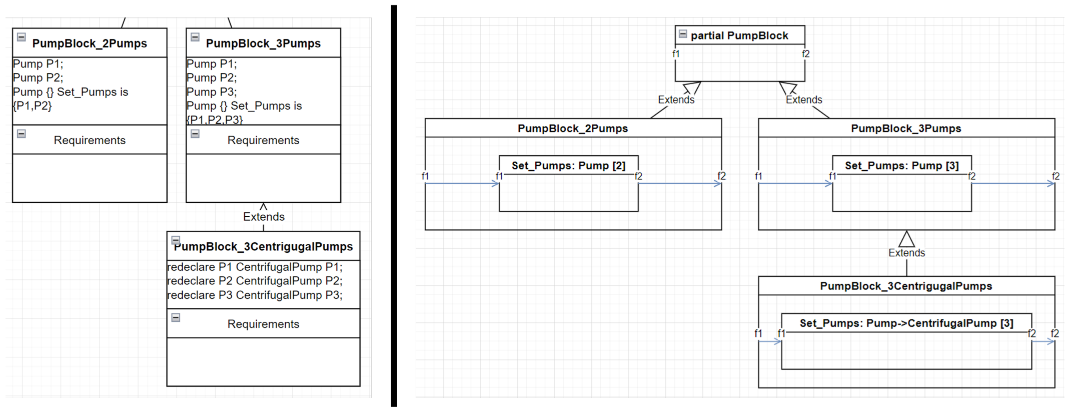


Figure 3.2.17: New definition of the class CentrifugalPump (Inheritance diagram on the left, Internal Structure diagram on the right)

The classes to be modified are then created, inheriting from the original class. If the three-pump architecture needs to be tested with the new definition of CentrifugalPump, a redeclaration will be made for that architecture. On the Internal Structure diagram, the redeclaration is marked by an arrow from the old class to the new one (Figure 3.2.18).



PumpBlock_2Pumps
Pump P1;
Pump P2;
Pump {} Set_Pumps is {P1,P2}
Requirements
PumpBlock_3Pumps
Pump P1;
Pump P2;
Pump P3;
Pump {} Set_Pumps is {P1,P2,P3}
Requirements
Extends
PumpBlock_3CentrigugalPumps
redeclare P1 CentrifugalPump P1;
redeclare P2 CentrifugalPump P2;
redeclare P3 CentrifugalPump P3;
Requirements
partial PumpBlock
f1
f2
Extends
Extends
PumpBlock_2Pumps
Set_Pumps: Pump [2]
PumpBlock_3Pumps
Set_Pumps: Pump [3]
Extends
PumpBlock_3CentrigugalPumps
Set_Pumps: Pump->CentrifugalPump [3]

Figure 3.2.18: Redeclaration of the class PumpBlock_3CentrigugalPumps (Inheritance diagram on the left, Internal Structure diagram on the right)

The model could be refined in other ways. For example, new classes could be created to model the internal structure of the pump. Indeed, the method diagrams can easily be refined according to the needs at the time.

3.2.4 Instance diagrams

Inheritance and internal structure diagrams are made to constitute a library of components that will be used. No instantiation is done yet. The instance diagram will instantiate the desired components. It must conform to the architecture contained in the internal structure diagram and must define all constants defined in the inheritance diagram. In addition, the individual parts mentioned in the internal structure diagram must be linked to the created instance.

The diagram starts with an empty space representing the place where the model is going to be made. The instances are then created (Figure 3.2.20). Only the attributes that need to be defined are written (i.e. what is not an external variable).

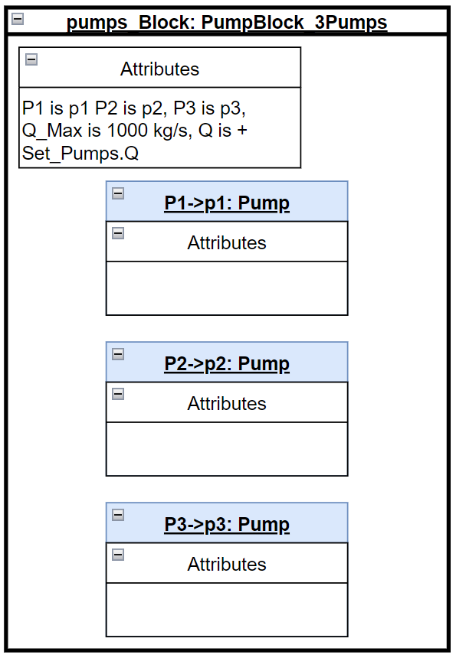

pumps_Block: PumpBlock_3Pumps
Attributes
P1 is p1 P2 is p2, P3 is p3, Q_Max is 1000 kg/s, Q is + Set_Pumps.Q
P1->p1: Pump
Attributes
P2->p2: Pump
Attributes
P3->p3: Pump
Attributes

Figure 3.2.19: Details of the PumpBlock instance in the Instance diagram

Instance names start with a lower case letter and are underlined. If the created instance corresponds to an individual part of the internal structure diagram, the name of the individual part is mentioned followed by an arrow and the name of the instance ("P1 -> p1: Pump"). This is also mentioned in the attributes of the parent class ("P1 is p1").

For pumps, no attributes need to be defined because all its attributes are external. The instance of PumpBlock_3Pumps has different attributes to set. QMax is a constant, it needs a value. Q is not external, it needs a value. It can be defined as the sum of the mass flows of the three pumps. This is a modelling choice. It could have been an external variable that would have been defined from the behaviour model via the bindings mechanism.

The CRML translation of Figure 3.2.20 is shown in Figure 3.2.21. The smallest components are instantiated first, then the parent class and so on.

```
Pump p1 is new Pump();
Pump p2 is new Pump();
Pump p3 is new Pump();

PumpBlock_3Pumps pumps_Block is new PumpBlock_3Pumps(P1 is p1,P2 is p2,P3 is p3,Q_Max is 1000kg/s, Q is + Set_Pumps.Q);
```

Figure 3.2.20: Translation into CRML of Figure 3.2.20

A possible Instance diagram with the chosen architecture (three pumps in parallel and two heat exchangers in parallel) is shown Figure 3.2.21.

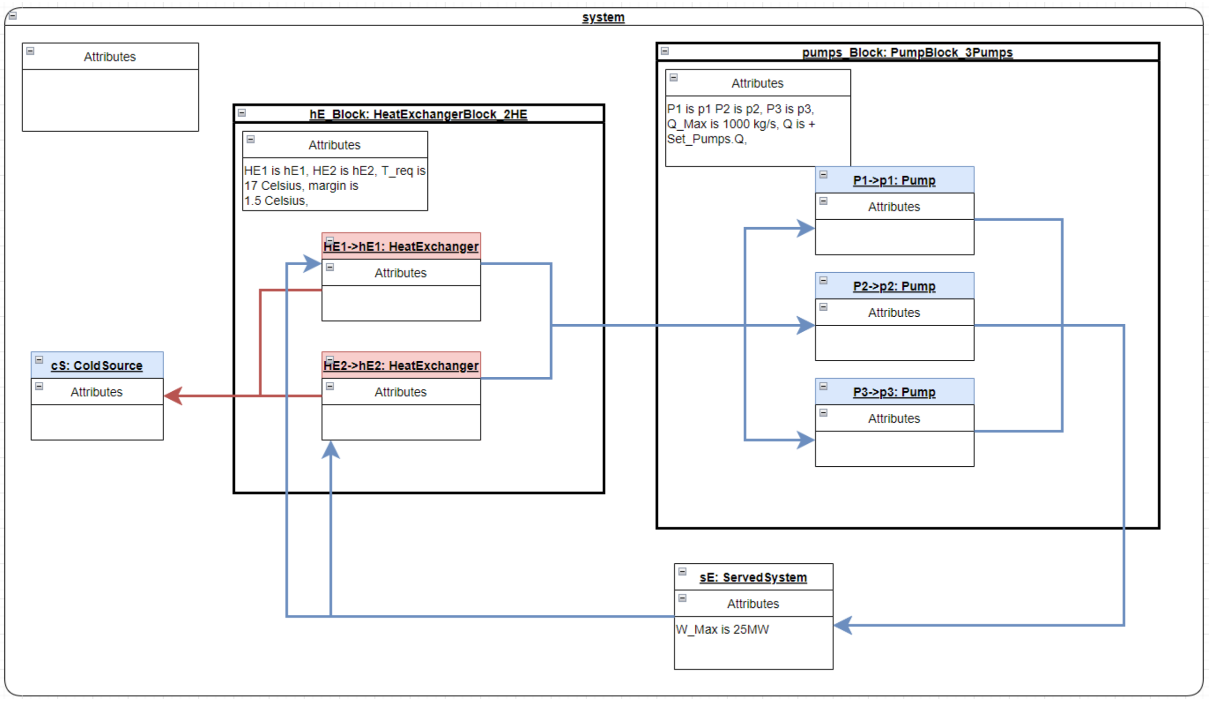


Figure 3.2.21: Instance diagram of the ICS

Physical requirements and CRML arrows

As mentioned earlier in the report, interfaces between components are also part of the method and it attempts to go further on this point and give not only a visual aid to the reader, but also the possibility to assign a requirement to it. A blue arrow on Figure 3.2.21, which at first glance might mean a fluid interface for the reader, could now be a defined fluid transfer with requirements to be verified such as mass conservation. The idea is to associate a requirement with a connector between pieces of equipment anywhere in the instance diagram.

This type of link with an associated requirement is called a CRML arrow in this method. The example of a CRML arrow representing a fluid interface will be studied here, but the main aim was to see if it was possible to implement CRML arrows in general in CRML and possibly in Modelica later on. This object is not part of the UML/SysML syntax. This arrow has to be drawn on the instance diagram but the definition of the class is on the inheritance diagram.

As CRML is an object-oriented language, a class is created for the CRML arrow representing the fluid exchange, called FluidLink. This class is created in the inheritance diagram as it is a class definition.

The first example of a requirement for the FluidLink will be the conservation of mass flow between two pieces of equipment. This will be an arrow pointing from equipment 1 to equipment 2. What is needed (to verify the conservation of mass flow rate) is the mass flow of the two components. In Modelica/ThermoSysPro, the mass flow rate can be defined as negative. Therefore, one must be careful and use the absolute value function. The requirement that will be verified is the following:

$$during\ TestPeriod\ ensure||Q_1| - |Q_2|| < \epsilon$$

with Q_i the mass flow rate of the equipment i, ϵ is a margin and $TestPeriod$ a boolean which both must be defined.

Graphically, the FluidLink is shown in Figure 3.2.22. Its translation into CRML is shown Figure 3.2.23.

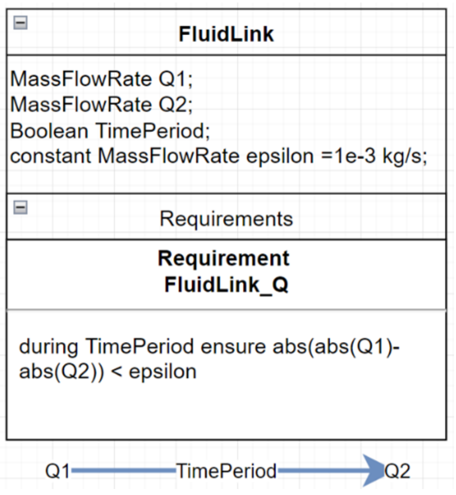


Figure 3.2.22: Implementation of the FluidLink in the Inheritance diagram

```
class FluidLink is {
    MassFlowRate Q1;
    MassFlowRate Q2;
    Boolean TimePeriod;
    constant MassFlowRate epsilon = 1e-3 kg/s;

    Requirement FluidLink_Q is during TimePeriod ensure abs(abs(Q1)- abs(Q2))<epsilon;
};
```

Figure 3.2.23: Implementation of the FluidLink in CRML

Presented as such with only mass flow conservation, the FluidLink could connect more than two pieces of equipment as mass flow rate is an extensive property. However, other requirements were tested and in particular, pressure requirements along the CRML arrow. What was verified is that between two components, the pressure cannot increase. Indeed, it is either equal (because nothing is modelled between these components in the behavioural model) or smaller at equipment 2 (because the diagram is a simplified view of the behavioural model and therefore not all pressure losses are modelled on the diagram). Thus, FluidLink can only connect two components.

The constraint that FluidLink can only connect two components is problematic when the components are in parallel (i.e. when the flow splits). In order to overcome this problem, fluid merges and splitters must be introduced. Splitters and merges are defined according to the number of inputs or outputs they have. If it has two inputs, it is an instance of Merger2. If it has three outputs, it is an instance of Splitter3. An example of PumpBlock_3Pumps is shown in Figure 4.1.1. If necessary, a requirement for mass conservation in the splitter or merger can be defined.

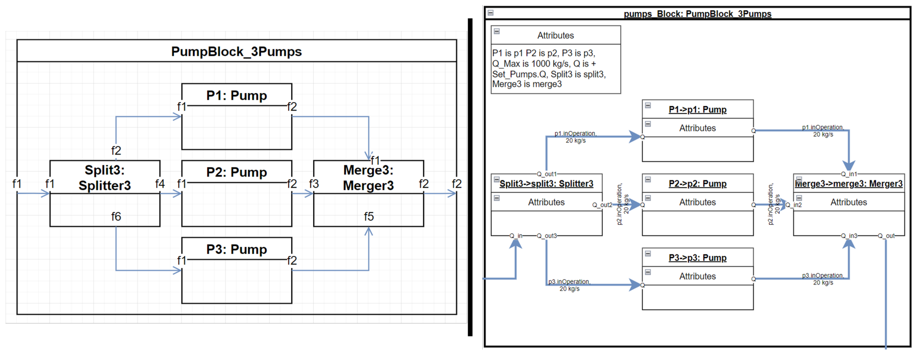


Figure 3.2.24: Comparison between the Internal Structure Diagram and the Instance diagram for components in parallel

In conclusion, introducing FluidLink (and thus splitters and mergers) tends to change the different diagrams. Indeed, new external variables and classes need to be declared in the Inheritance diagram. Splitters and mergers need to be added to the Internal Structure diagram and to the Instance diagrams. However, this exercise has shown the possibility of implementing CRML arrows in the diagram. Indeed, in the CRML language, it is just a class instantiated several times.

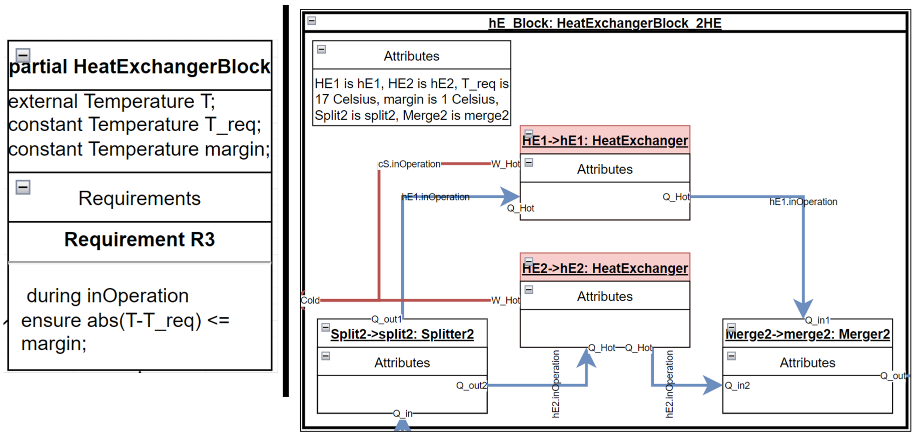
The complete diagrams of this method applied to the ICS example (with the FluidLink and thus the splitters and mergers) are presented in Appendix A.

3.3 Simulation to evaluation requirements generated by the graphical

For the purposes of this presentation, the simulation will be carried out. As shown in Figure 3.1.1, this method is part of a more complex requirements verification process. The other tools in the diagram are not explained in this report as they were not the focus of the master thesis. We therefore assume that following the realisation of the previous graphical method, CRML code has been generated and linked to the behavioural and architectural models by the bindings.

To simplify the presentation, the focus will be on requirement 3 of the ICS : Req. 3: "*In normal operation, the temperature at the outlet of the heat exchangers shall be close to*

17°C within a ±1.5°C margin with a confidence rate of x%.". This requirement relates to the HeatExchangerBlock and is shown in Figure 3.3.1.


```
partial HeatExchangerBlock
external Temperature T;
constant Temperature T_req;
constant Temperature margin;
Requirements
Requirement R3
during inOperation
ensure abs(T-T_req) <=
margin;
```

```
hE_Block: HeatExchangerBlock_2HE
Attributes
HE1 is hE1, HE2 is hE2, T_req is
17 Celsius, margin is 1 Celsius,
Split2 is split2, Merge2 is merge2
```

Figure 3.3.1: Implementation of the requirement 3 in the Inheritance and Instance diagrams

The simulation of the verification model is then carried out on the Dymola software (Dymola 2018, with DASSL solver) for a duration of t=450s. At t=200s, the served systems drop in temperature, which will cause the overall system temperature to drop. The ICS must then regulate the temperature back to 17°C. The requirement verifies the correct operation of this feature. The requirement will be checked between t=60s and t=360s.

The result of a simulation gives the temperature and the status of the requirement (`true`, `false`, `undecided` or `undefined`). A first simulation gives the results shown in Figure 3.3.2.

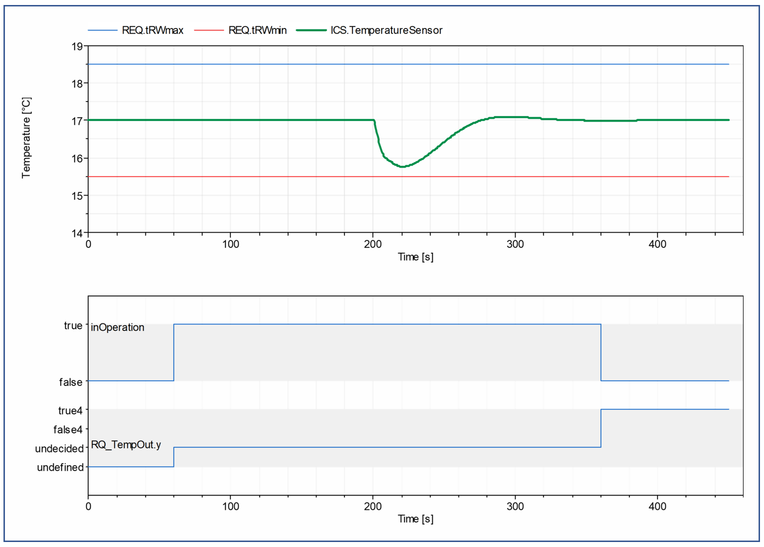


Figure 3.3.2: Simulation results for the variation of temperature at the outlet of the HeatExchangerBlock for a small variation of the temperature in the ServedSystem

It can be seen that the temperature at the outlet of the exchangers varies but never exceeds the threshold of 1.5°C that has been imposed. The requirement is first `Undefined` because the test time period has not started yet. The requirement then changes to `Undecided` because we cannot yet conclude on its outcome. At the end of the test period, the temperature has remained within the imposed range, so the requirement is verified.

Let's now impose a higher temperature variation on the systems served. The model is therefore more restrictive.

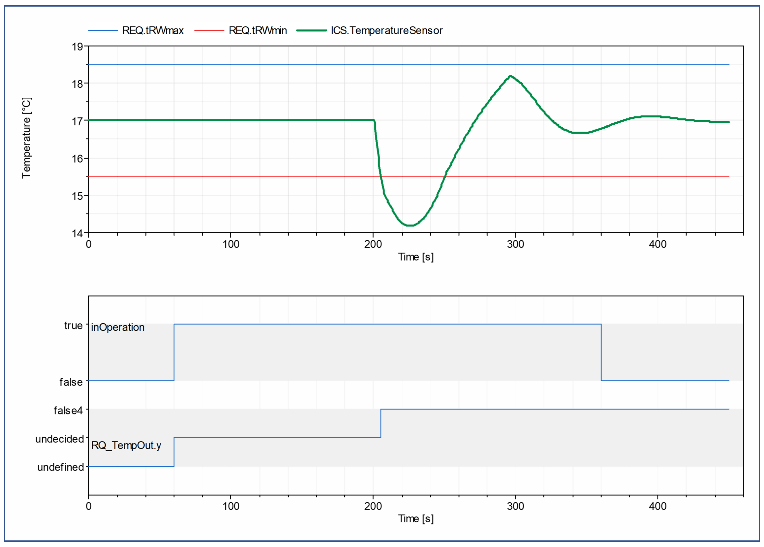


Figure 3.3.3: Simulation results for the variation of temperature at the outlet of the HeatExchangerBlock for a small variation of the temperature in the ServedSystem

Similarly, the requirement is first `Undefined` and then `Undecided` in Figure 3.3.3, but the temperature change is such that the system has not been able to adapt quickly enough, and the temperature at the HeatExchanger outlet has fallen below the imposed threshold. The requirement is then immediately violated (before reaching the end of the test period as it is known to have been violated at least once).

To go further, we can ask ourselves for example why the choice of two exchangers in our model. Would one have been sufficient with the considered cold source? The results are shown in Figure 3.3.4.

We then modify the behavioural model to have only one exchanger: we immediately realize that the system cannot maintain the water at 17°C with the initial temperature of the systems served. The requirement is therefore violated from the start of the test.

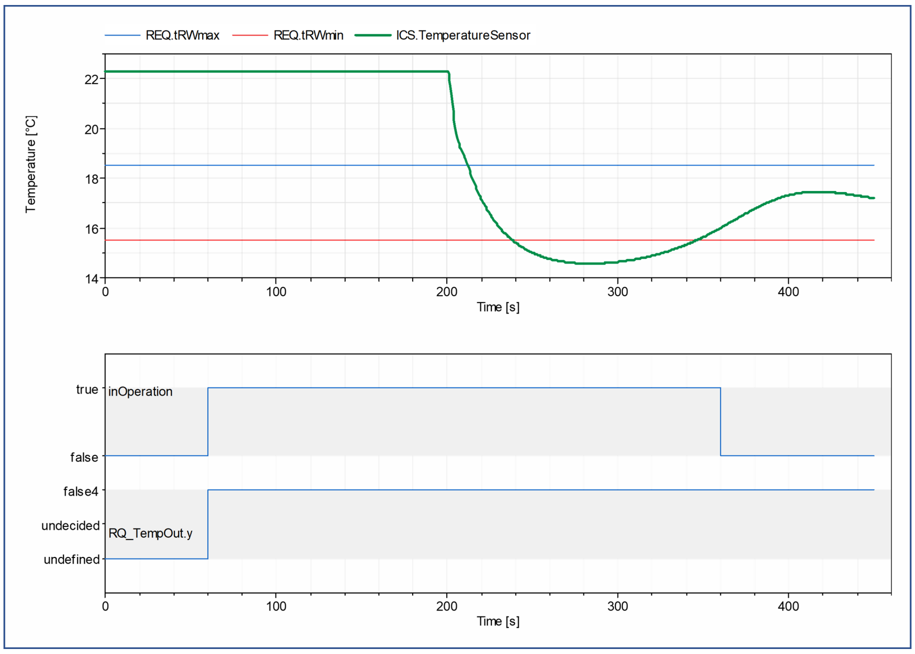


Figure 3.3.4: Simulation results for the variation of temperature at the outlet of the HeatExchangerBlock with only one HeatExchanger for a variation of the temperature in the ServedSystem

Chapter 4

Application for the design of SMR

This part deals with the application of the new graphical method to the case of the secondary circuit of an SMR. First, a presentation of the SMR and of the non-electric applications of it will be made. The second part will deal with the application of the method itself by taking an industrial point of view on the problem.

4.1 SMR and non-electric applications

SMRs are small reactors with a power of between 300 and 700 MWe. EDF's SMR project is called NUWARD™ (for NuclearForward) and the objectives are clear: create a modular reactor adaptable to different environments or uses. The NUWARD™ project focuses on small SMRs functioning by pair of two units with a power of 170 MWe produced per reactor.Launched in September 2019, the NUWARD™ SMR project brings together several stakeholders: EDF, TechnicAtome, Framatome, Tractebel and the CEA. Mainly dedicated to export, the concept aims to supply isolated regions or energy-intensive industrial sites, for example in emerging countries, with decarbonised electricity and to replace ageing coal fired-plants. To have quite rapidly a concrete showcase of the product, the aim of the NUWARD™ project is to have the first of a kind (FoaK) plant in service by 2030 in France.

Several innovations are key to this project, such as a particularly compact steam generator, a primary circuit entirely integrated into the vessel and passive safety systems (i.e. without the need for external energy).

Further studies are being carried out to investigate the use of the SMR not only in power plants but also for uses such as hydrogen production, district heating or water desalination. This involves using part of the thermal power supplied by the reactor for so-called ”non-electric” applications. These kinds of hybridization scenarios are in particular studied in parallel of

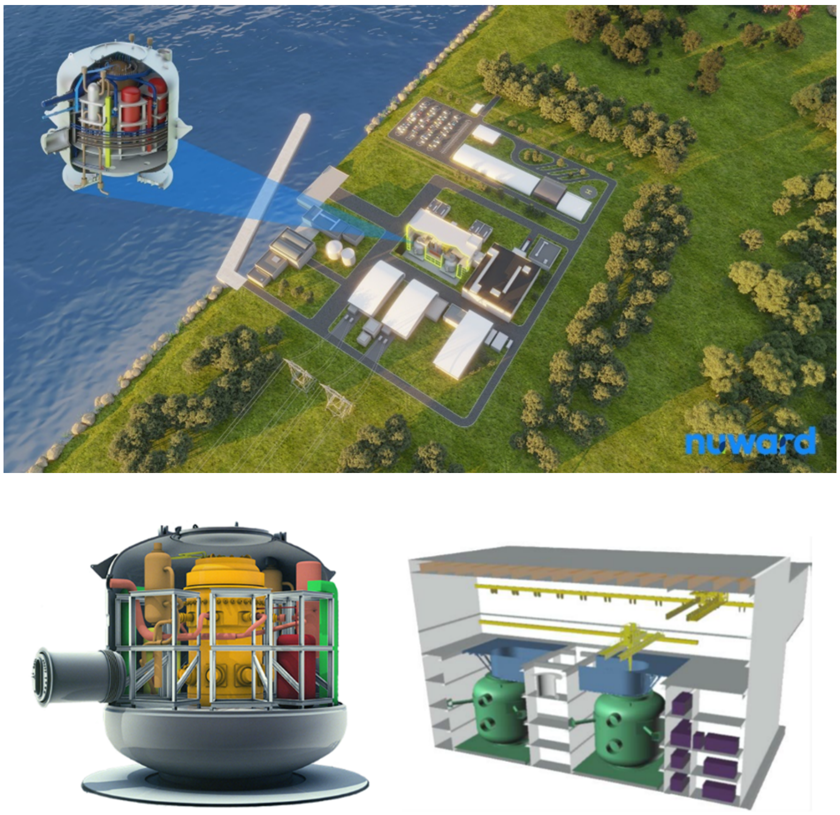


Figure 4.1.1: Presentation of NUWARD™ project and its design principles, a 340MWe SMR plant with two independent reactors (170MWe each) housed in a single nuclear building

NUWARD™ project through the European project TANDEM.

4.2 Application of the graphical method to the secondary circuit of an SMR

Now let us consider the case of the secondary circuit of an SMR and try to apply the new graphical design method. If one assumes that we are in the place of the engineer who must deal with the design of the secondary loop, then we can easily understand that he is facing different requirements. These requirements may come from the strategic specifications of the project, they may be specific to the equipment considered, the safety standards he has to consider or even the requirements linked to his own expertise that he wants to respect (based for instance on some performance criteria). These requirements may have been set prior to his involvement in the project and choices will therefore have been made and can no longer be discussed.

The question which then arises is how he should go about managing these different requirements and how they could guide him in making appropriate design choices. One solution may therefore be the requirements model as suggested in the previous sections. More concretely let suppose for the suite of the example that the engineer inherits from previous project discussions the following set of high-level requirements:

- **NUW-001**: This requirement is provided by the strategic specification of the NUWARD™ project. This is the primary objective of the reactor: *in normal operation and at 100% of the nominal power, the plant shall produce at least 340 MWe when the cold source is at 15°C.* NUWARD™ consists of two production units. Thus each reactor must provide at least 170MWe.
- **NTI-RQ-010**: This requirement relates to the thermal power at the interface between the primary and secondary circuit. *In normal operation and at 100% of the nominal power, the thermal power must be greater than 540MWth.*
- **NTI-RQ-010**: *NUWARD™ uses superheated steam at the outlet of the steam generators*, this is an additional requirement to be verified
- **NTI-RQ-005**: *This requirement relates to the flow rate and temperature range at the inlet of the steam generators in the secondary circuit.* It is interesting to look at what happens at the steam generator levels because it is the characteristics of the Rankine cycle that we want to represent for the secondary cycle.
- **NUW-010**: This requirement addresses the responsiveness of the system during transients. *In between 100% of the nominal power and the technical minimum, the ramps must be at least 3% of the nominal power per minute* (sources ENTSOe and EUR)
- **NUW-049**: This requirement addresses the hybrid nature of the SMR: *up to xx% of total thermal power can be allocated to cogeneration.*

As explained in 2.2, those requirements need to be written as technical requirements to be more precise (i.e. unambiguous and measurable) and be integrated in the graphical design method previously presented.

4.2.1 First design of the secondary loop as a “classical” Rankine cycle

We are interested in designing the secondary cycle of an SMR plant. The operation of a power plant is always relatively similar, a (slightly modified) Rankine cycle is always the basis, and we will use this cycle to start our modelling. The Rankine cycle has four successive phases [7]. The liquid is vaporised in the vaporisation phase (by heat exchange with the primary circuit). The steam then passes through the turbines during the expansion. The gas then passes through a condenser where it is liquefied (by heat exchange with the cooling circuit). The liquid is then reheated and pressurised during the compression phase so that the cycle can be started again (see Figure 4.2.1). These four phases (**compression** 1-2, **vaporisation**

2-3, **expansion** 3-4 and **liquefaction** 4-1) constitute the main principle of the secondary loop from a thermodynamical point of view. In order to represent the power plant unit in a comprehensible way and to represent the different interfaces of the secondary circuit, the primary circuit, the cooling circuit and the transformer will also be represented in the power plant unit but will not be detailed in the modelling. They will only be black boxes with respect to the secondary circuit and will be modelled as interfaces requirements.

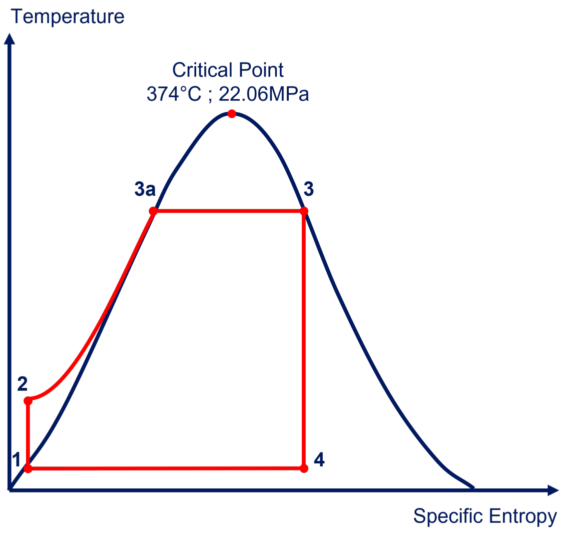


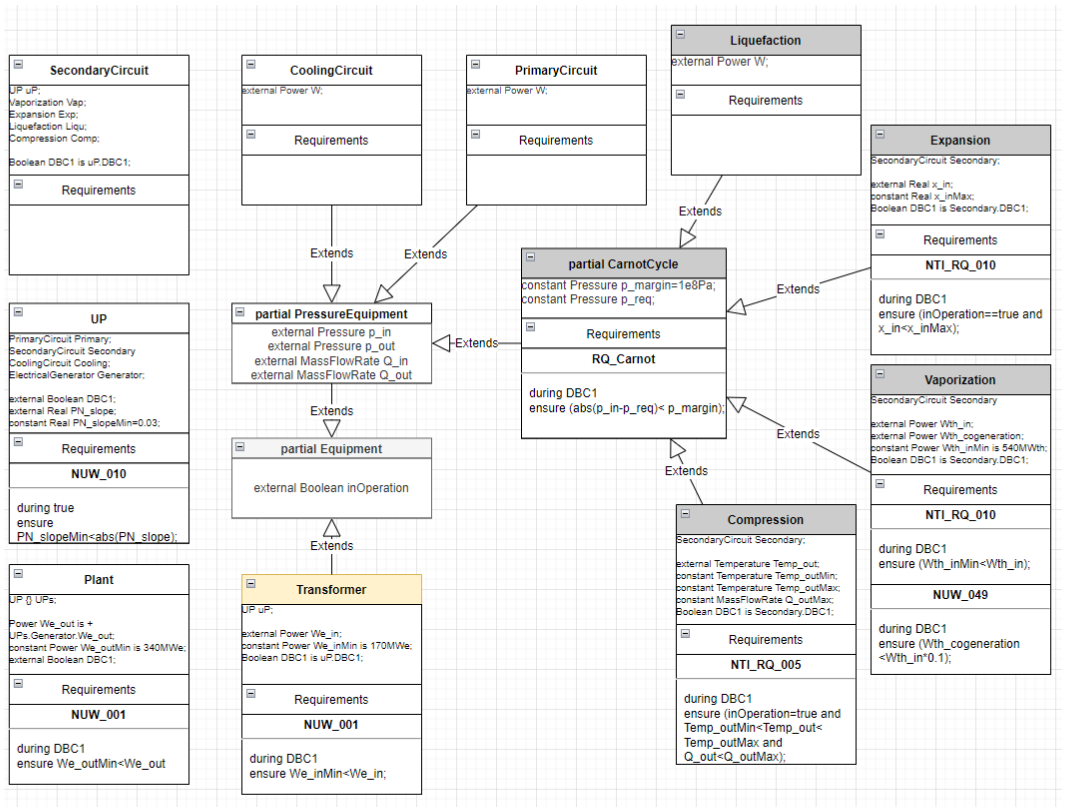
Figure 4.2.1: Simple Rankine Cycle for the secondary circuit of a nuclear power plant

By considering a precise thermodynamic cycle, requirements for the operating point of the Rankine cycle we want to represent have to be considered: for example, point 3 has a pressure p_3. Thus, a new requirement stating that "Requirement RQ_Rankine: *the pressure at the outlet of the vaporisation shall be about* p_3 *in normal operation*" can be defined (and so on for the other phases).

With the different requirements coming from the strategic specifications as well as the desired operating points for the Rankine cycle, it is possible to apply the previous method to the system. We opt for a view based on the physical phenomena of the Rankine cycle. This is a modelling choice. We therefore have the different phases to represent within classes, each with requirements from the specifications. The requirements concerning the Rankine Cycle (RQ_Rankine) have to be verified by the four phases: the notion of inheritance can be used.

Results of the method can then be seen on Figures 4.2.2, 4.2.3 and 4.2.4 or in Appendix C. The different classes are defined with the needed variables for the requirements defined previously.

Other interfaces than fluid interfaces may be seen on Figure 4.2.3 and 4.2.4. The red CRML



- **NUW-001**: *in normal operation and at 100% of the nominal power, the plant shall produce at least 340 MWe when the cold source is at 15°C*
- **NTI-RQ-010**: *In normal operation and at 100% of the nominal power, the thermal power must be greater than 540MWth*
- **NTI-RQ-010**: *NUWARD™ uses superheated steam at the outlet of the steam generators*
- **NTI-RQ-005**: *This requirement relates to the flow rate and temperature range at the inlet of the steam generators in the secondary circuit*
- **NUW-010**: *In between 100% of the nominal power and the technical minimum, the ramps must be at least 3% of the nominal power per minute (sources ENTSOe and EUR)*
- **RQ_Carnot**: *in normal operation and at 100% of the nominal power, the interface shall have a pressure of xPa, within a yPa margin*

Figure 4.2.2: Instance diagram for the first detailed phase of the method with the requirements to be verified

arrows represent heat interfaces, with requirement regarding thermal power conservation. The yellow CRML arrow represent an electrical interface with a requirement regarding the electrical power conservation.

4.2.2 Refined design to improve the efficiency of the Rankine cycle

We want to go further in designing the secondary loop in particular to improve its energy efficient. This is done by refining the graphical diagrams previously modeled. If we focus on the power generation part of the model (i.e. the expansion phase), we can see in the specifications that in reality the Rankine cycle taken previously is too simple and does not represent the reality

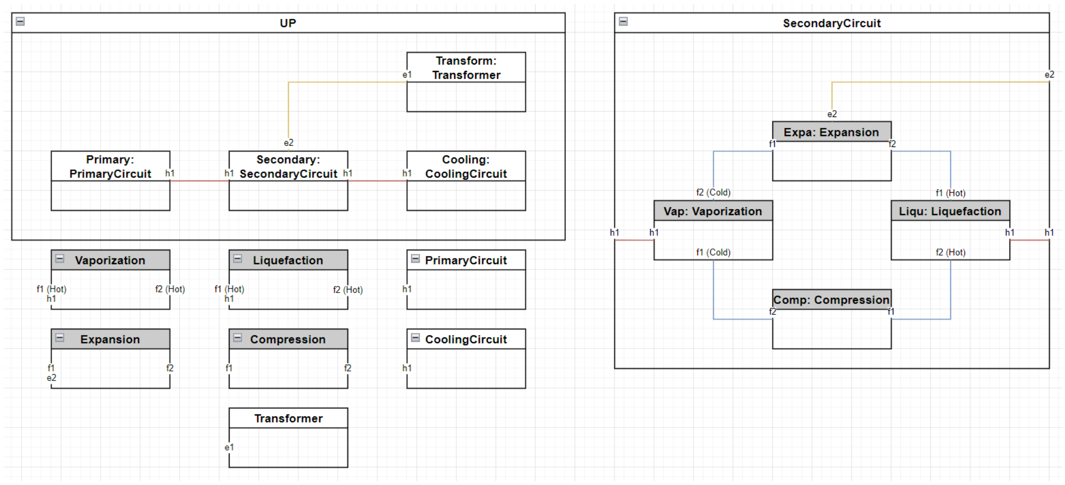


Figure 4.2.3: InternalStructure diagram for the first detailed phase of the method

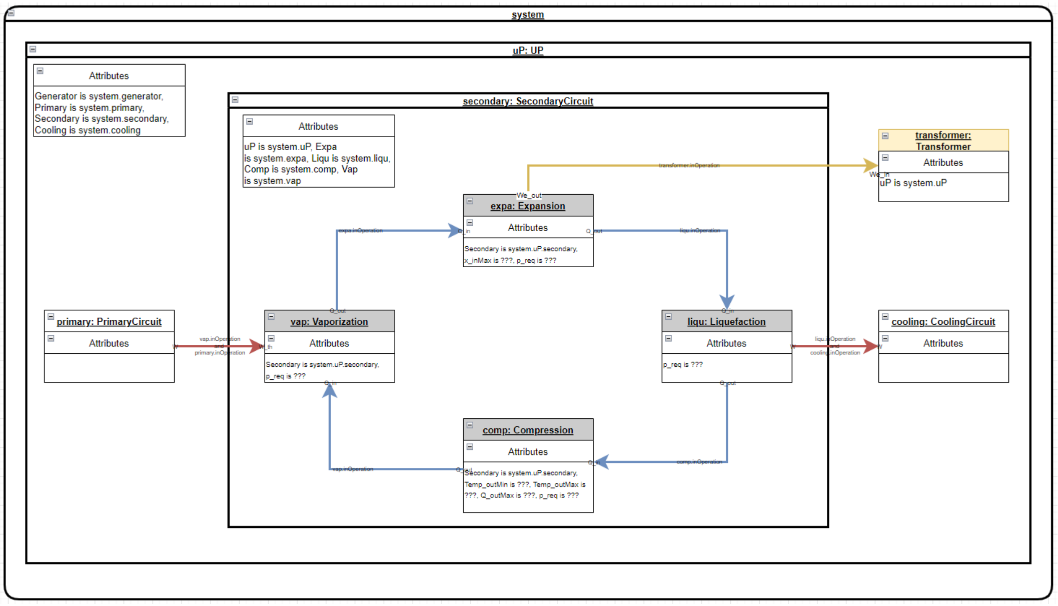

system
uP: UP
Attributes
Generator is system.generator, Primary is system.primary, Secondary is system.secondary, Cooling is system.cooling
secondary: SecondaryCircuit
uP is system.uP, Expa is system.expa, Liqu is system.liqu, Comp is system.comp, Vap is system.vap
transformer: Transformer
uP is system.uP
expa: Expansion
Secondary is system.uP.secondary, x_inMax is ???, p_req is ???
primary: PrimaryCircuit
vap: Vaporization
Secondary is system.uP.secondary, p_req is ???
liqu: Liquefaction
p_req is ???
cooling: CoolingCircuit
comp: Compression
Secondary is system.uP.secondary, Temp_outMin is ???, Temp_outMax is ???, Q_outMax is ???, p_req is ???

Figure 4.2.4: Instance diagram for the first detailed phase of the method

of the circuit. Indeed, the NUWARD™ document also stipulate the use of superheated steam as well as a high- and low-pressure turbines with a Moisture Separator Reheater (MSR) between the two. A diagram of the Rankine cycle is given in Figure 4.2.5. The expansion is considered from point 3 to 6.

New operating points are envisaged for the refined Rankine cycle with new requirements. New classes are created for the HighPressureTurbine block, LowPressureTurbine block, MSR and ElectricalGenerator.

Liquefaction and vaporisation can also be refined and considered as heat exchangers to the primary circuit and the cooling circuit. The HeatExchanger is a new system component and has associated requirements. For example, "Requirement *v_MAX*: *the fluid velocity at the*

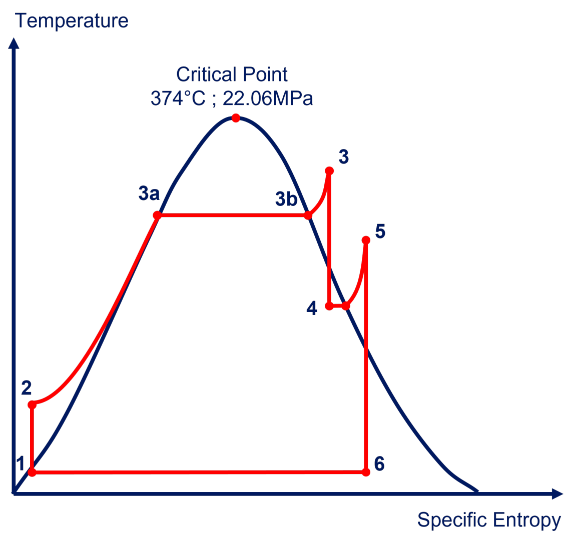


Figure 4.2.5: Refined Rankine Cycle for the secondary circuit of a nuclear power plant

HeatExchanger inlets must not be greater than a maximum to ensure optimal exchange between the cold and hot circuits in the HeatExchanger."

The components we focused on (Expansion, Liquefaction and Vaporisation) which were elementary in stage 1 are now refined with their internal structure (Figure 4.2.6).

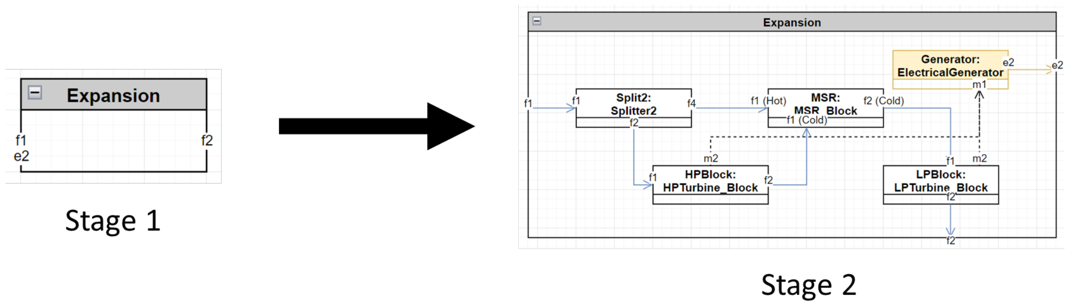


Figure 4.2.6: Difference between stage 1 and 2 for the Expansion phase. Expansion was an elementary component phase 1, in phase 2 its internal structure is given

Refined diagrams can be seen in Figure 4.2.7 and Appendix D for better readability. The important thing is not to understand each box but to see the possibilities left by the graphical method, such as adding requirements/classes as you go along. The added classes during the refine phases could, in the future, be found in library of components with already written requirements.

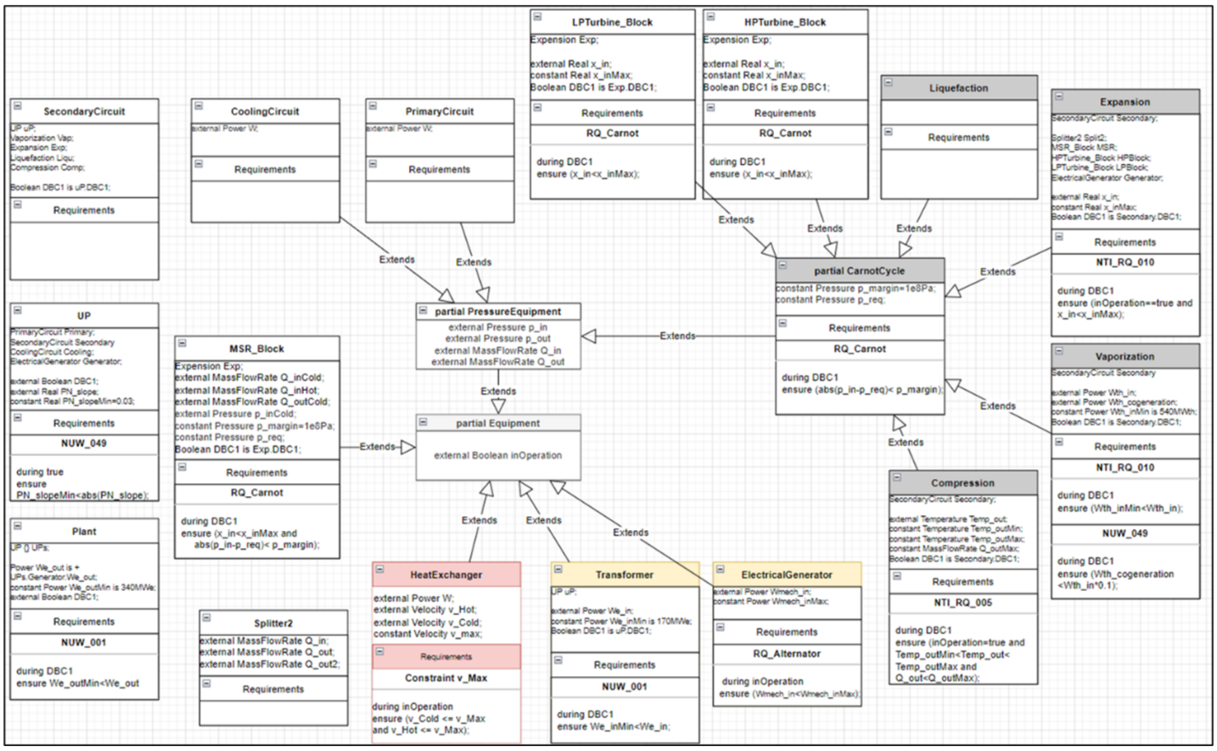


Figure 4.2.7: Instance diagram for the second detail phase of the method

4.2.3 Further refinement by defining concrete components and their associated requirements

Further refinement of the models would be possible. Blocks with high- and low-pressure turbines could now be specified with the internal structure, thus the number of turbines in each. A new turbine class would then be created with associated requirements. These could be related to the maximum inlet temperature (constraint due to blades and surface treatments), or a maximum pressure at the turbine inlet (to avoid admitting wet steam) as shown in Figure 4.2.8.

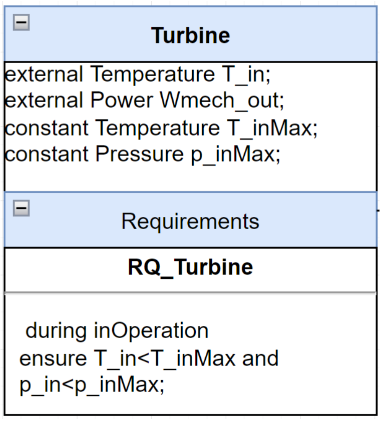

Turbine
external Temperature T_in;
external Power Wmech_out;
constant Temperature T_inMax;
constant Pressure p_inMax;
Requirements
RQ_Turbine
during inOperation
ensure T_in<T_inMax and
p_in<p_inMax;

Figure 4.2.8: Turbine definition in a refined Inheritance diagram

Similarly, the HeatExchangers defined above are in fact a condenser and a steam generator (Figure 4.2.9). These new components would also have associated requirements defined by the domain engineer.

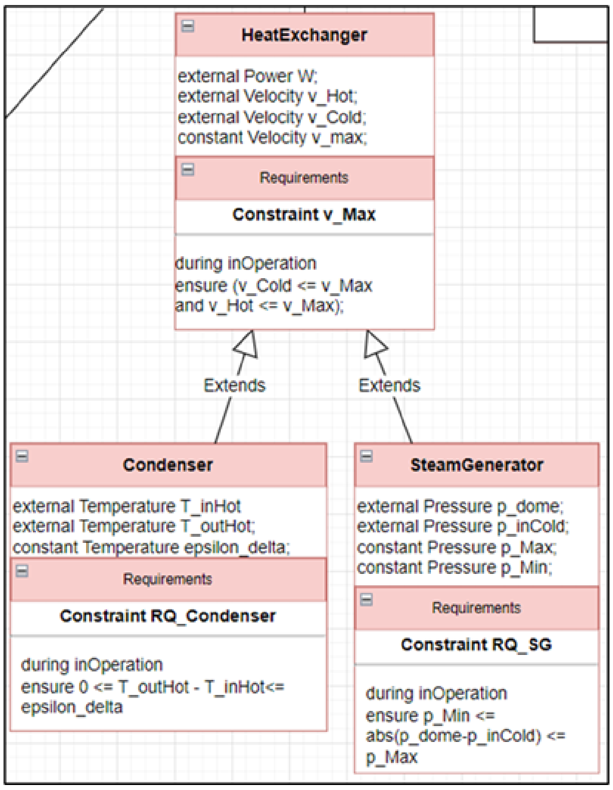


Figure 4.2.9: Condenser and SteamGenerator definition in a refined Inheritance diagram

Having the precise number of sub-components in a component (as is the case for the number of turbines now in the HPTurbineBlock) shows that requirements modelling is already relatively advanced in the process and that a possible solution is already considered.

Refined diagrams can be seen in Appendix E for better readability.

Chapter 5

Conclusions

The verification of requirements is crucial for their successful completion to handle the complexity of large projects such as ME-CPS. It can be automated through the creation of a requirements model which enables the formalization and hence the computation of requirements. The purpose of this requirements model is to structure the different requirements of the project, whether they are taken from the high-level specifications, from previous design choices or from the different engineering teams or stakeholders involved in the project. Having this model allows a rigourous justification of the design choices made by the engineer and therefore a clear tracking of knowledge and decisions. It is made to help the engineer in its design choices.

The CRML language was developed within EDF R&D in order to provide a rigorous framework for the requirements model and its verification. In order to facilitate the adoption of the model and CRML, a graphical method was developed during the master thesis. This method aimed at organising the requirements of ME-CPS as well as managing different possible architectures for the system in order to verify the requirements in different scenarios. It is composed of three different diagrams to create classes and instances in CRML. It is inspired from different systems engineering concepts found in the literature. However, unlike the UML and SysML graphical syntaxes, this new graphical method eventually allows the creation of formal CRML code directly from the diagrams and hence the possibility to simulate them.

This report presents the development of the graphical method and its application to two examples: an intermediate cooling system used in a power plant (ICS) and the secondary circuit of a small modular reactor (SMR). Those two examples of ME-CPS have been processed by the graphical method in order to organise the CRML code of the requirements model. These results are encouraging in the appropriation of the CRML language as most of the diagrams (the inheritance and internal structure diagrams) could be capitalized in the form of libraries and be reused as templates for applying the method to other use cases. Its application to ME-CPS at large seems to be feasible since it was thought as complementary to the larger framework drafted in a thesis done at EDF R&D [1].

5.1 Future Work

However, the graphical method is part of a more global project. Only the beginning of the method is presented here. To consolidate it, it should be tested on other energy systems, in particular on different scenarios of use of the SMR in the framework of the TANDEM project. Its applicability could also be tested at the operation stage, for instance to monitor energy systems or to help the on-site reception of plant components (the verification process could use in that case some on-site measurements to represent the system states instead of a behavioural model as shown in the report).

Moreover, concerning the method itself, the notion of interface could be extended with the addition of the notion of Assumed/Guarantee (A/G) contracts [1]. Finally, the tools present in the project continue to be developed in the EMBrACE project. To rigorously test the semantics of the graphical method, a tool allowing the creation of CRML code directly from the diagrams need to be prototyped. Similarly, the finalisation of the CRML compiler to Modelica is necessary and planned as a priority for the upcoming months. Discussions with the various partners (Linköping University, SAAB, etc.) are continuing, in particular for promoting CRML as a future standard either in the Modelica association or in the OMG group. To that aim a first integration of CRML into SysML V2 has been prototyped by Siemens Belgium.

5.2 Final Words

In conclusion, the results shown in this report and the work carried out during this internship are encouraging with regard to the adoption of requirements model in an engineering process. The different objectives defined at the beginning of the project for the method have been verified, but other need to be explored in order to represent the full complexity of engineering projects such as the design of an SMR secondary circuit.

Bibliography

[1] Azzouzi, Elmehdi. "Multi-Faceted Modelling of Multi-Energy Systems : Stakeholders Coordination". PhD These. Université Paris-Saclay, Jan. 2021. URL: https://tel.archives-ouvertes.fr/tel-03166509.

[2] Bouskela, Daniel. "CRML specification". In: Aug. 2022. URL: https://www.embrace-project.org/specification/CRML.pdf.

[3] Bouskela, Daniel. "Modeling Architecture for the Verification of Requirements - MODRIO deliverable D2.1.1". In: H-P1C-2014-15188-EN, EDF - Internal document. Nov. 2015.

[4] Bouskela, Daniel and Jardin, Audrey. "ETL: A new temporal language for the verification of cyber-physical systems". In: Apr. 2018, pp. 1–8. DOI: 10.1109/SYSCON.2018.8369502.

[5] Bouskela, Daniel, Nguyen, Thuy, and Jardin, Audrey. "Towards a rigorous approach for verifying cyber-physical systems against requirements". In: Oct. 2015, pp. 250–255. DOI: 10.1109/EPEC.2015.7379958.

[6] *Composite structure diagrams*. Aug. 2021. URL: https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-composite-structure.

[7] Dincer, Ibrahim. *Comprehensive Energy Systems*. 2018. ISBN: 978-0-12-814925-6.

[8] EL Hefni, Baligh and Bouskela, Daniel. *Modeling and Simulation of Thermal Power Plants with ThermoSysPro: A Theoretical Introduction and a Practical Guide*. Jan. 2019. ISBN: 978-3-030-05104-4. DOI: 10.1007/978-3-030-05105-1.

[9] Ellervee, Peeter and Tammemäe, Kalle. "VHDL – VHSIC Hardware Description Language". In: URL: http://mini.li.ttu.ee/~lrv/IAY0040/vhdl-basics.pdf.

[10] Jardin, Audrey et al. "CRML a Language for Verifying Realistic Dynamic Requirements". In: URL: https://github.com/lenaRB/crml-compiler/blob/main/resources/crml_tutorial/tutorial_slides.pdf.

[11] Nguyen, Thuy. "Formal Requirements and Constraints Modelling in FORM-L for the Engineering of Complex Socio-Technical Systems". In: *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. Dec. 2019, pp. 123–132. DOI: 10.1109/REW.2019.00027.

[12] *OMG Unified Modeling Language*$^{TM}(OMG - UML)Version2.5$. Object Management Group. 2015, pp. 181–195. URL: http://www.omg.org/spec/UML/2.5/.

[13] Schamai, Wladimir. "Modelica Modeling Language (ModelicaML)". In: *Technical Report. Linköping University Electronic Press*. 2009.

[14] Smith, James. "What About Ada? The State of the Technology in 2003". In: (July 2003). DOI: 10.1184/R1/6585905.v1. URL: https://kilthub.cmu.edu/articles/report/What_About_Ada_The_State_of_the_Technology_in_2003/6585905.

[15] Tao, Ran, Song, Xijie, and Ye, Changliang. "Pumped Storage Technology, Reversible Pump Turbines and Their Importance in Power Grids". In: *Water* 14.21 (Nov. 2022), p. 3569. ISSN: 2073-4441. DOI: 10.3390/w14213569. URL: http://dx.doi.org/10.3390/w14213569.

Appendix - Contents

A Diagrams of the graphical method applied to the ICS 52

B CRML code examples 56

C Diagrams of the graphical method applied to the first design of the secondary loop of the SMR 58

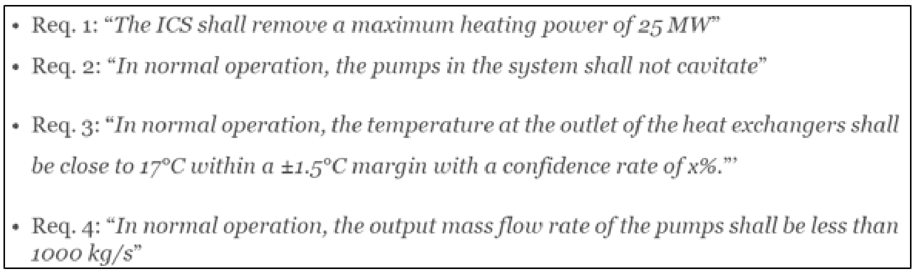
D Diagrams of the graphical method applied to the refined design of the secondary loop of the SMR 62

E Diagrams of the graphical method applied to the most refined design of the secondary loop of the SMR 66

Appendix A

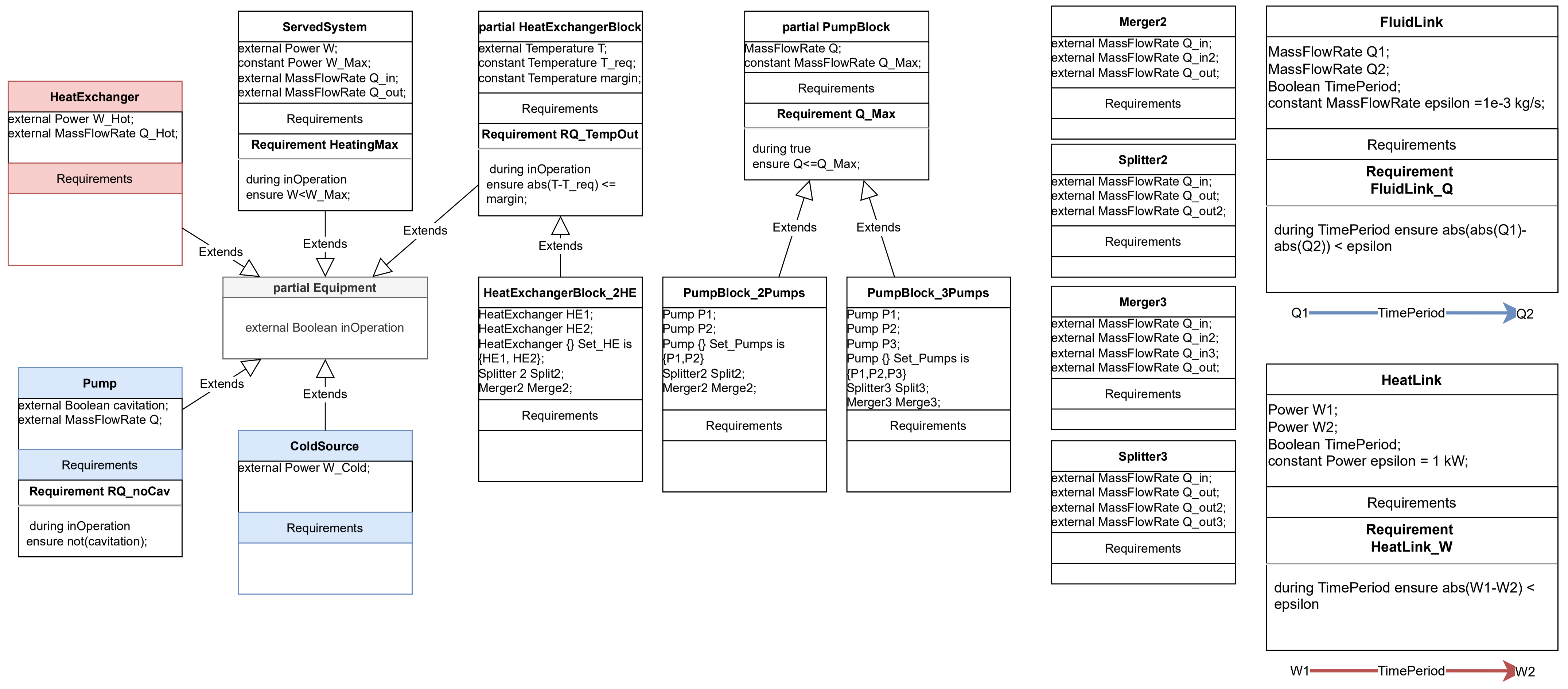
Diagrams of the graphical method applied to the ICS

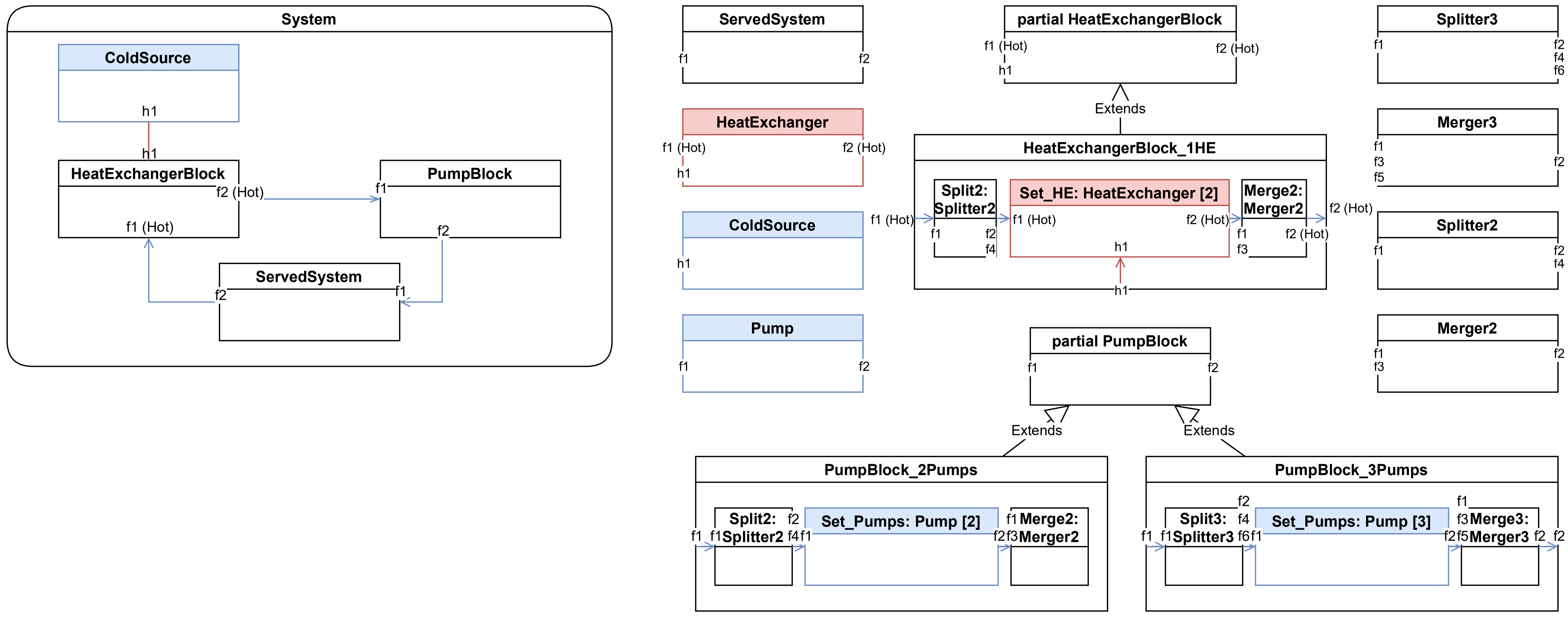
This appendix presents the three diagrams for the ICS example discussed in Chapter 3 that were produced when applying the new graphical method. The inheritance, internal structure and instance diagrams are presented. The requirements to be verified are shown Figure A.0.1.



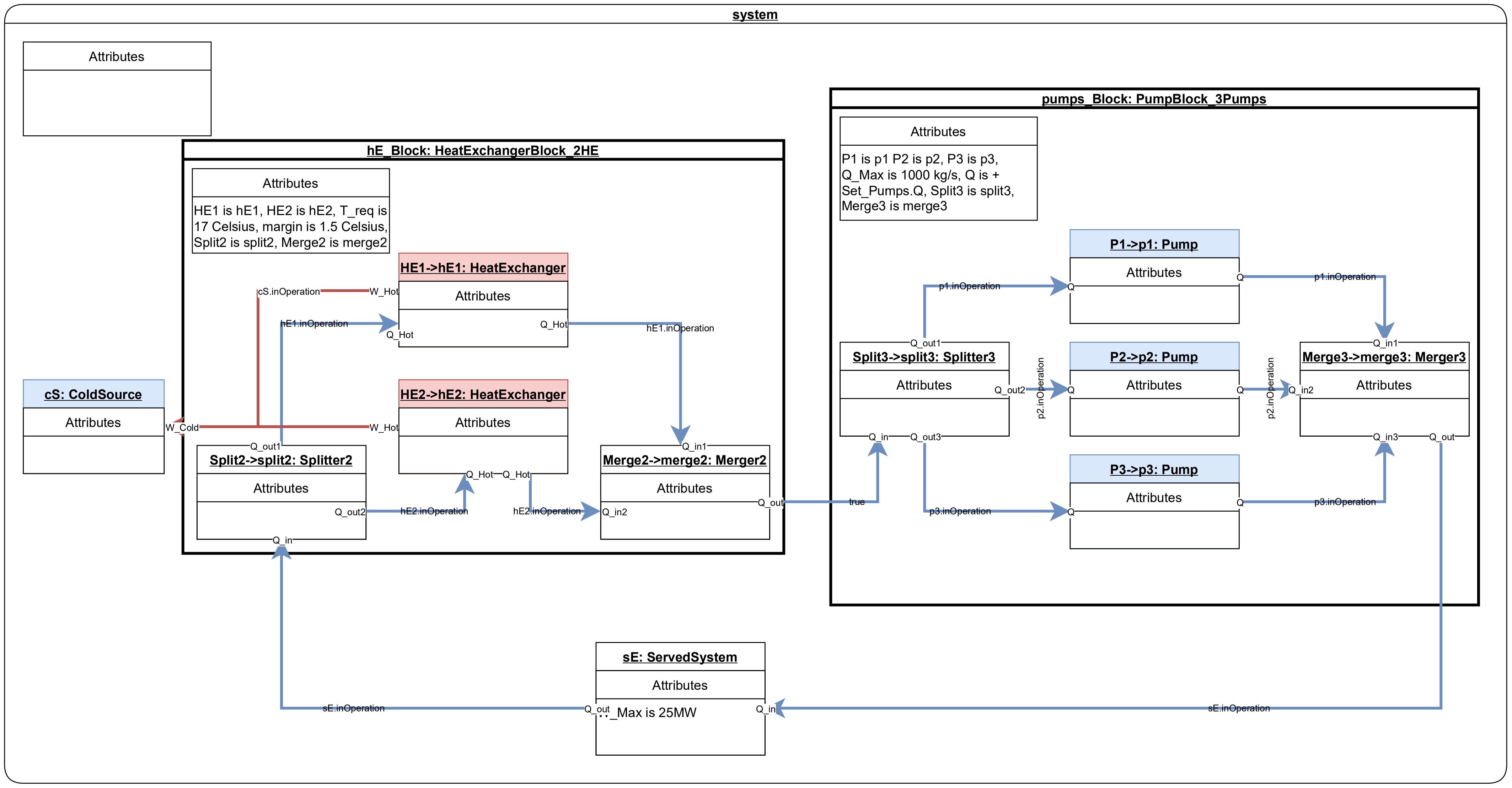
- Req. 1: "*The ICS shall remove a maximum heating power of 25 MW*"
- Req. 2: "*In normal operation, the pumps in the system shall not cavitate*"
- Req. 3: "*In normal operation, the temperature at the outlet of the heat exchangers shall be close to 17°C within a ±1.5°C margin with a confidence rate of x%.*"'
- Req. 4: "*In normal operation, the output mass flow rate of the pumps shall be less than 1000 kg/s*"

Figure A.0.1: Requirements to be verified by the ICS


HeatExchanger
external Power W_Hot;
external MassFlowRate Q_Hot;
Requirements
ServedSystem
external Power W;
constant Power W_Max;
external MassFlowRate Q_in;
external MassFlowRate Q_out;
Requirements
Requirement HeatingMax
during inOperation
ensure W<W_Max;
partial HeatExchangerBlock
external Temperature T;
constant Temperature T_req;
constant Temperature margin;
Requirements
Requirement RQ_TempOut
during inOperation
ensure abs(T-T_req) <=
margin;
partial PumpBlock
MassFlowRate Q;
constant MassFlowRate Q_Max;
Requirements
Requirement Q_Max
during true
ensure Q<=Q_Max;
Merger2
external MassFlowRate Q_in;
external MassFlowRate Q_in2;
external MassFlowRate Q_out;
Requirements
Splitter2
external MassFlowRate Q_in;
external MassFlowRate Q_out;
external MassFlowRate Q_out2;
Requirements
Merger3
external MassFlowRate Q_in;
external MassFlowRate Q_in2;
external MassFlowRate Q_in3;
external MassFlowRate Q_out;
Requirements
Splitter3
external MassFlowRate Q_in;
external MassFlowRate Q_out;
external MassFlowRate Q_out2;
external MassFlowRate Q_out3;
Requirements
FluidLink
MassFlowRate Q1;
MassFlowRate Q2;
Boolean TimePeriod;
constant MassFlowRate epsilon =1e-3 kg/s;
Requirements
Requirement FluidLink_Q
during TimePeriod ensure abs(abs(Q1)-abs(Q2)) < epsilon
Q1
TimePeriod
Q2
HeatLink
Power W1;
Power W2;
Boolean TimePeriod;
constant Power epsilon = 1 kW;
Requirements
Requirement HeatLink_W
during TimePeriod ensure abs(W1-W2) < epsilon
W1
TimePeriod
W2
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Extends
partial Equipment
external Boolean inOperation
HeatExchangerBlock_2HE
HeatExchanger HE1;
HeatExchanger HE2;
HeatExchanger {} Set_HE is
{HE1, HE2};
Splitter 2 Split2;
Merger2 Merge2;
Requirements
PumpBlock_2Pumps
Pump P1;
Pump P2;
Pump {} Set_Pumps is
{P1,P2}
Splitter2 Split2;
Merger2 Merge2;
Requirements
PumpBlock_3Pumps
Pump P1;
Pump P2;
Pump P3;
Pump {} Set_Pumps is
{P1,P2,P3}
Splitter3 Split3;
Merger3 Merge3;
Requirements
Pump
external Boolean cavitation;
external MassFlowRate Q;
Requirements
Requirement RQ_noCav
during inOperation
ensure not(cavitation);
ColdSource
external Power W_Cold;
Requirements



System
ColdSource
h1
HeatExchangerBlock
f1 (Hot)
f2 (Hot)
PumpBlock
f1
f2
ServedSystem
ServedSystem
HeatExchanger
ColdSource
Pump
partial HeatExchangerBlock
Extends
HeatExchangerBlock_1HE
Split2: Splitter2
Set_HE: HeatExchanger [2]
Merge2: Merger2
partial PumpBlock
PumpBlock_2Pumps
Set_Pumps: Pump [2]
PumpBlock_3Pumps
Split3: Splitter3
Set_Pumps: Pump [3]
Merge3: Merger3
Splitter3
Merger3
Splitter2
Merger2
f3
f4
f5
f6


system
Attributes
cS: ColdSource
Attributes
W_Cold
hE_Block: HeatExchangerBlock_2HE
Attributes
HE1 is hE1, HE2 is hE2, T_req is 17 Celsius, margin is 1.5 Celsius, Split2 is split2, Merge2 is merge2
cS.inOperation
W_Hot
hE1.inOperation
HE1->hE1: HeatExchanger
Attributes
Q_Hot
Q_Hot
hE1.inOperation
HE2->hE2: HeatExchanger
Attributes
W_Hot
Q_Hot
Q_Hot
Q_out1
Split2->split2: Splitter2
Attributes
Q_out2
hE2.inOperation
hE2.inOperation
Q_in
Q_in1
Merge2->merge2: Merger2
Attributes
Q_in2
Q_out
true
pumps_Block: PumpBlock_3Pumps
Attributes
P1 is p1 P2 is p2, P3 is p3, Q_Max is 1000 kg/s, Q is + Set_Pumps.Q, Split3 is split3, Merge3 is merge3
P1->p1: Pump
Attributes
Q
Q
p1.inOperation
p1.inOperation
Q_out1
Split3->split3: Splitter3
Attributes
Q_out2
p2.inOperation
Q_in
Q_out3
P2->p2: Pump
Attributes
Q
Q
p2.inOperation
Q_in1
Merge3->merge3: Merger3
Attributes
Q_in2
Q_in3
Q_out
P3->p3: Pump
Attributes
Q
Q
p3.inOperation
p3.inOperation
sE: ServedSystem
Attributes
Q_out
_Max is 25MW
Q_in
sE.inOperation
sE.inOperation

Appendix B

CRML code examples

This appendix presents examples of CRML code taken from the CRML tutorial made for the EMBrACE project presentation [10]. This is the latest version of CRML. It may differ from the one used in the diagrams and examples in this report.

CRML Expressions

- Expressions

```
[ [ type ] ident is ] [ value | external ] [; | ,]
```

- Comments

```
// This is a single-line comment

/* This is a
multiline comment */
```

CRML Variables

- Types
  - `Real var;`
  - `Boolean var;`
  - `Integer var;`
  - `String pathToH; // Path to the observation operator H`
- Variability
  - Variables are function of time
  - Constant variables: variables assumed to be constant in time
    `constant TypeVar var`
  - Parameters: variables assumed to be constant during simulation time, but values could be different from one simulation to another
    `parameter TypeVar var`
- Value definition
  - Either directly via an expression: `TypeVar var is …;`
  - Or via another model: `TypeVar var is external;`

Definition of Classes

```
class Pump is {
    String ident;
    Boolean isStarted is external;
    Real temperature is external;

    // R1: While the system is in operation, the pump must not be started more than twice.
    Requirement R1 is
        'during' system.inOperation 'check count' (pump.isStarted 'becomes true') '<=' 2;
    // R2: At least one hour must separate two consecutive pump startups.
    Requirement R2 is
        'after' pump.isStarted 'for' 1*h 'check count' (pump.isStarted 'becomes true') '==' 0;
    // R3: While the pump is in operation (i.e. started), its temperature must always stay below 50°C.
    Requirement R3 is
        'during' pump.isStarted 'ensure' pump.temperature < 50*degC;
    // R4: While the system is in operation, after the pump temperature rises above 40 °C,
    // the temperature must not stay above for a duration of more than 1 mn cumulated over the next 15 mn.
    Requirement R4 is
        'during' system.inOperation 'after' pump.temperature > 40*degC 'for' 15*mn
            'check duration' (pump.temperature > 40*degC) '<' 1*mn;
};
```

The 4 requirements can be encapsulated into a generic class « Pump »

When the system is defined, each requirement is automatically instanciated for each pump

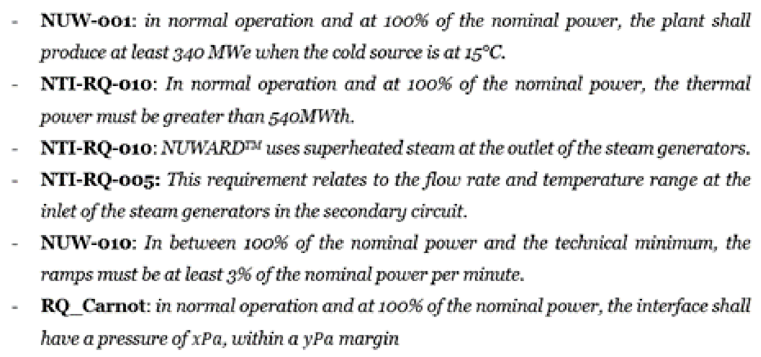
```
class System is {
    Pump {} pumps;
    Boolean inOperation is external;
};

System system is System{ Pump (ident = "PO1"),  Pump (ident = "PO2"), Pump (ident = "PO3") };
```

Appendix C

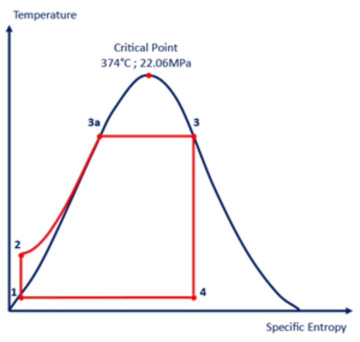
Diagrams of the graphical method applied to the first design of the secondary loop of the SMR

This appendix presents the three diagrams for the SMR example discussed in Section 4.2.1 that were produced when applying the new graphical method. The inheritance, internal structure and instance diagrams are presented. The requirements to be verified and the Carnot cycle used as a model are shown Figure C.0.1.



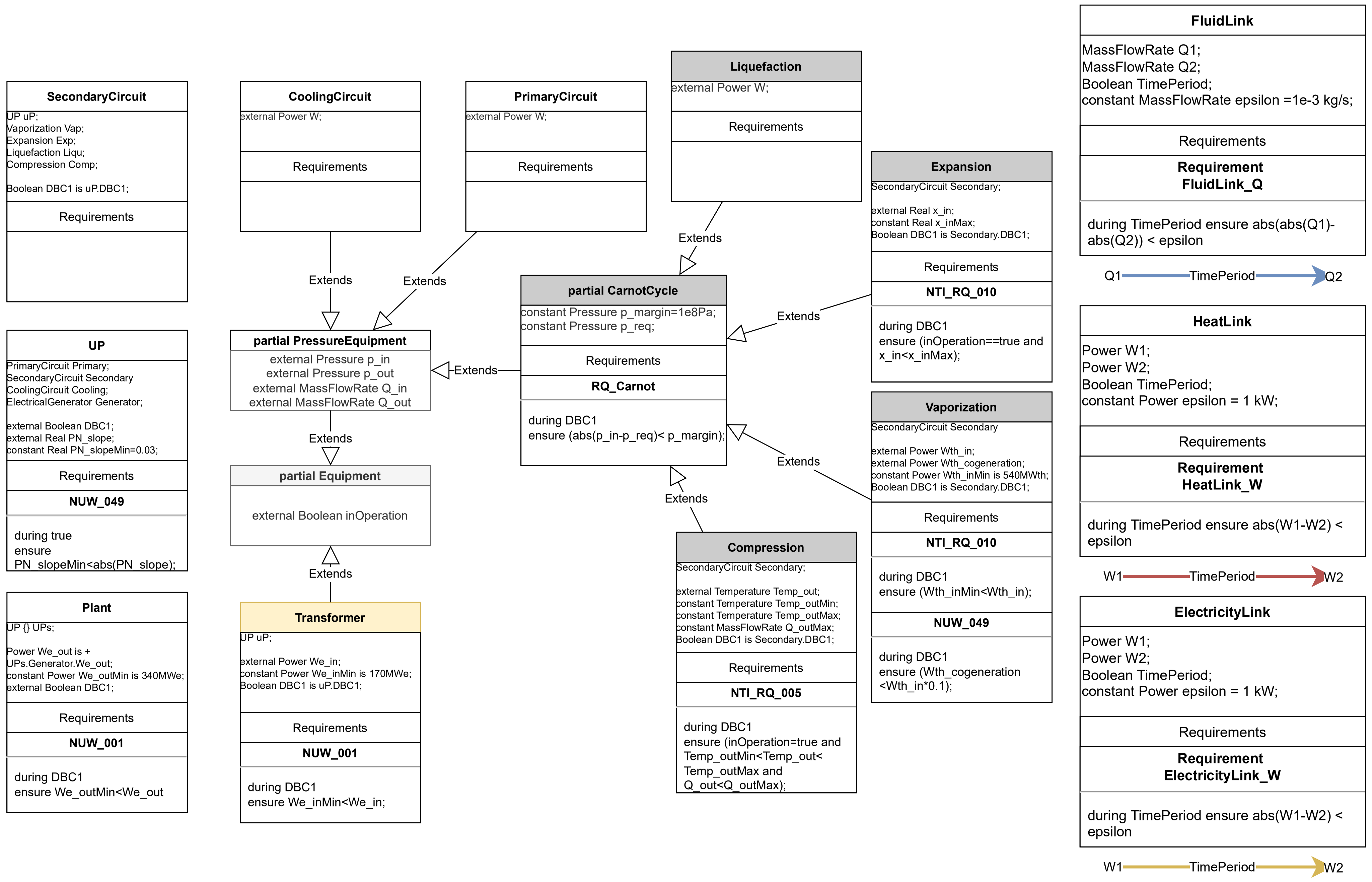
- **NUW-001**: *in normal operation and at 100% of the nominal power, the plant shall produce at least 340 MWe when the cold source is at 15°C.*
- **NTI-RQ-010**: *In normal operation and at 100% of the nominal power, the thermal power must be greater than 540MWth.*
- **NTI-RQ-010**: *NUWARD™ uses superheated steam at the outlet of the steam generators.*
- **NTI-RQ-005**: *This requirement relates to the flow rate and temperature range at the inlet of the steam generators in the secondary circuit.*
- **NUW-010**: *In between 100% of the nominal power and the technical minimum, the ramps must be at least 3% of the nominal power per minute.*
- **RQ_Carnot**: *in normal operation and at 100% of the nominal power, the interface shall have a pressure of xPa, within a yPa margin*

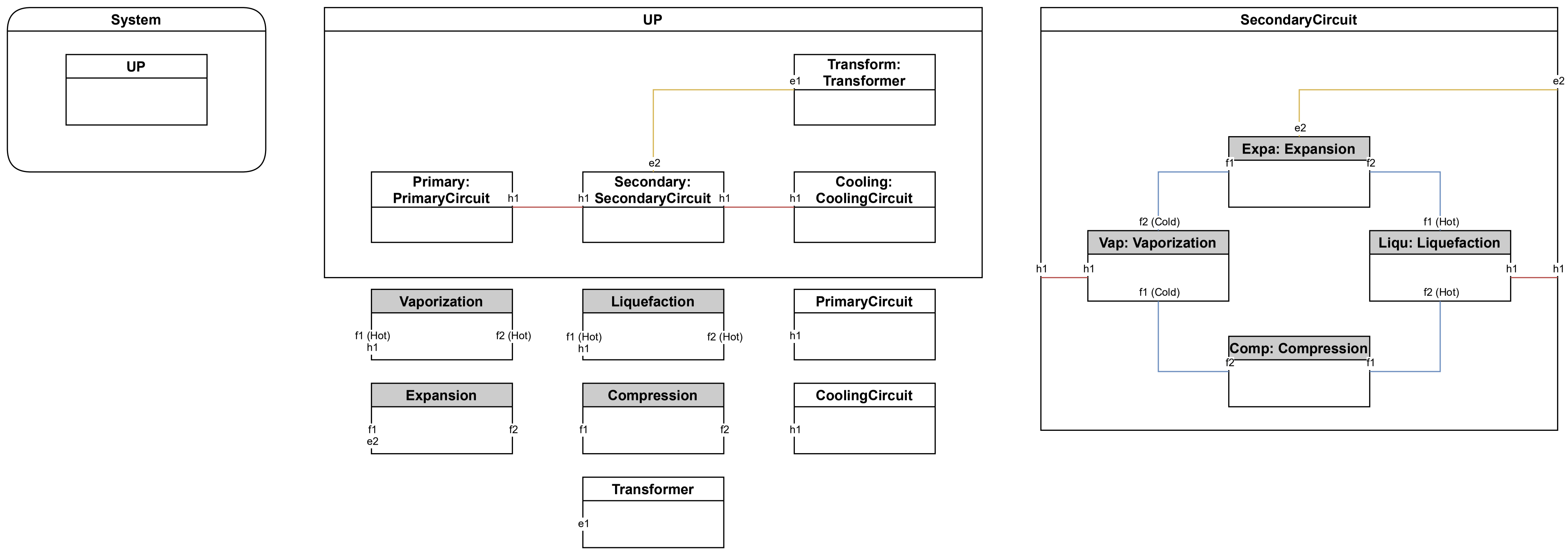
(a) Requirements to be verified by the first design of the secondary loop of the SMR



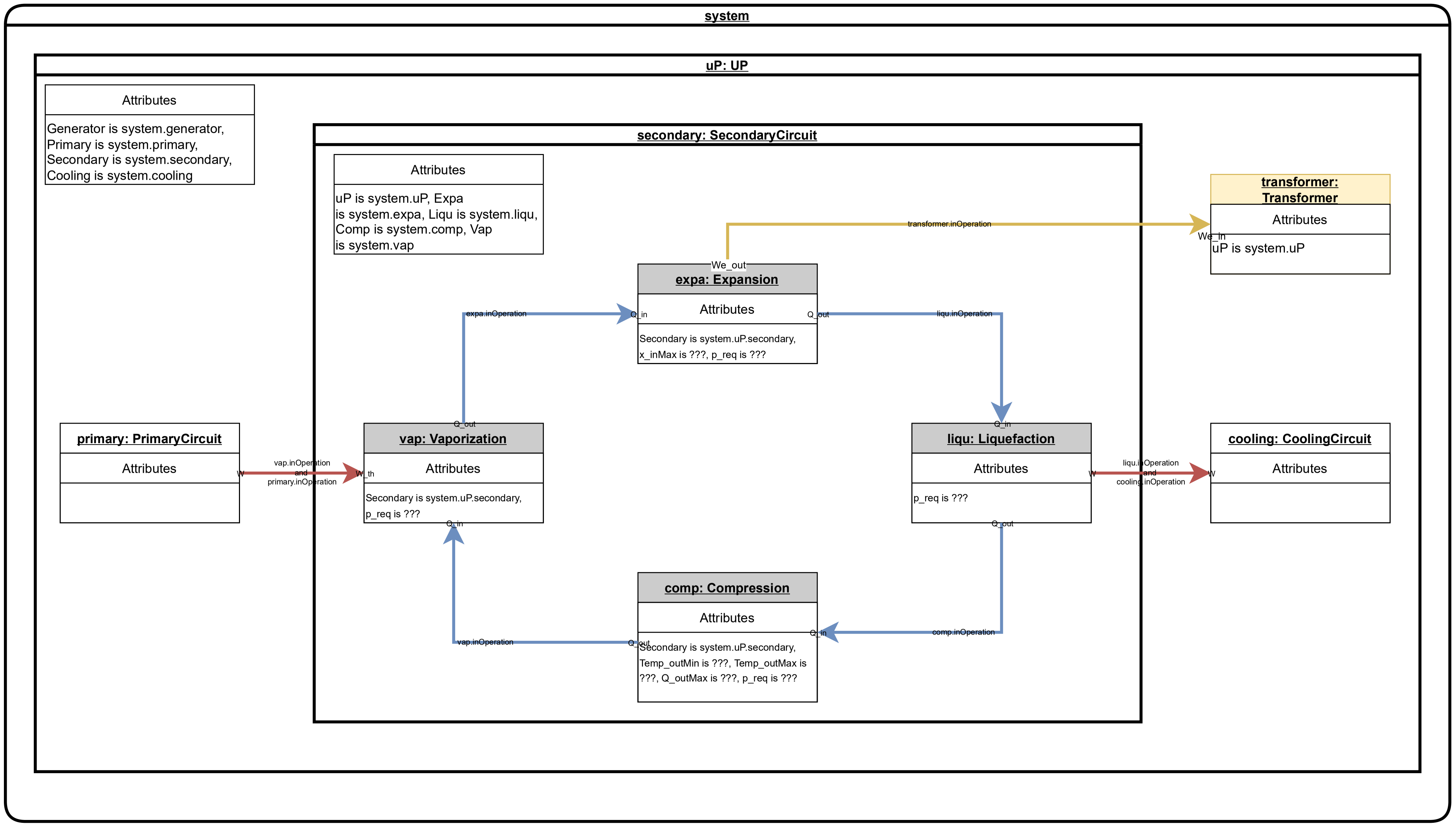
(b) Simple Carnot Cycle for the secondary circuit of a nuclear power plant

Figure C.0.1: Requirements and Carnot Cycle used as a base for the first design


SecondaryCircuit
UP uP;
Vaporization Vap;
Expansion Exp;
Liquefaction Liqu;
Compression Comp;
Boolean DBC1 is uP.DBC1;
Requirements
UP
PrimaryCircuit Primary;
SecondaryCircuit Secondary
CoolingCircuit Cooling;
ElectricalGenerator Generator;
external Boolean DBC1;
external Real PN_slope;
constant Real PN_slopeMin=0.03;
Requirements
NUW_049
during true
ensure
PN_slopeMin<abs(PN_slope);
Plant
UP {} UPs;
Power We_out is +
UPs.Generator.We_out;
constant Power We_outMin is 340MWe;
external Boolean DBC1;
Requirements
NUW_001
during DBC1
ensure We_outMin<We_out
CoolingCircuit
external Power W;
Requirements
PrimaryCircuit
external Power W;
Requirements
Liquefaction
external Power W;
Requirements
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Extends
partial PressureEquipment
external Pressure p_in
external Pressure p_out
external MassFlowRate Q_in
external MassFlowRate Q_out
partial Equipment
external Boolean inOperation
Transformer
UP uP;
external Power We_in;
constant Power We_inMin is 170MWe;
Boolean DBC1 is uP.DBC1;
Requirements
NUW_001
during DBC1
ensure We_inMin<We_in;
partial CarnotCycle
constant Pressure p_margin=1e8Pa;
constant Pressure p_req;
Requirements
RQ_Carnot
during DBC1
ensure (abs(p_in-p_req)< p_margin);
Expansion
SecondaryCircuit Secondary;
external Real x_in;
constant Real x_inMax;
Boolean DBC1 is Secondary.DBC1;
Requirements
NTI_RQ_010
during DBC1
ensure (inOperation==true and
x_in<x_inMax);
Vaporization
SecondaryCircuit Secondary
external Power Wth_in;
external Power Wth_cogeneration;
constant Power Wth_inMin is 540MWth;
Boolean DBC1 is Secondary.DBC1;
Requirements
NTI_RQ_010
during DBC1
ensure (Wth_inMin<Wth_in);
NUW_049
during DBC1
ensure (Wth_cogeneration
<Wth_in*0.1);
Compression
SecondaryCircuit Secondary;
external Temperature Temp_out;
constant Temperature Temp_outMin;
constant Temperature Temp_outMax;
constant MassFlowRate Q_outMax;
Boolean DBC1 is Secondary.DBC1;
Requirements
NTI_RQ_005
during DBC1
ensure (inOperation=true and
Temp_outMin<Temp_out<
Temp_outMax and
Q_out<Q_outMax);
FluidLink
MassFlowRate Q1;
MassFlowRate Q2;
Boolean TimePeriod;
constant MassFlowRate epsilon =1e-3 kg/s;
Requirements
Requirement
FluidLink_Q
during TimePeriod ensure abs(abs(Q1)-
abs(Q2)) < epsilon
Q1
TimePeriod
Q2
HeatLink
Power W1;
Power W2;
Boolean TimePeriod;
constant Power epsilon = 1 kW;
Requirements
Requirement
HeatLink_W
during TimePeriod ensure abs(W1-W2) <
epsilon
W1
TimePeriod
W2
ElectricityLink
Power W1;
Power W2;
Boolean TimePeriod;
constant Power epsilon = 1 kW;
Requirements
Requirement
ElectricityLink_W
during TimePeriod ensure abs(W1-W2) <
epsilon
W1
TimePeriod
W2



System
UP
UP
Transform: Transformer
e1
e2
Primary: PrimaryCircuit
h1
h1
Secondary: SecondaryCircuit
h1
h1
Cooling: CoolingCircuit
Vaporization
f1 (Hot)
h1
f2 (Hot)
Liquefaction
f1 (Hot)
h1
f2 (Hot)
PrimaryCircuit
h1
Expansion
f1
e2
f2
Compression
f1
f2
CoolingCircuit
h1
Transformer
e1
SecondaryCircuit
e2
e2
Expa: Expansion
f1
f2
f2 (Cold)
f1 (Hot)
Vap: Vaporization
Liqu: Liquefaction
h1
h1
h1
h1
f1 (Cold)
f2 (Hot)
Comp: Compression
f2
f1


system
uP: UP
Attributes
Generator is system.generator, Primary is system.primary, Secondary is system.secondary, Cooling is system.cooling
secondary: SecondaryCircuit
Attributes
uP is system.uP, Expa is system.expa, Liqu is system.liqu, Comp is system.comp, Vap is system.vap
transformer: Transformer
Attributes
We_in
uP is system.uP
transformer.inOperation
We_out
expa: Expansion
Attributes
Q_in
Q_out
Secondary is system.uP.secondary, x_inMax is ???, p_req is ???
expa.inOperation
liqu.inOperation
primary: PrimaryCircuit
Attributes
W
vap.inOperation and primary.inOperation
vap: Vaporization
Attributes
W_th
Q_out
Q_in
Secondary is system.uP.secondary, p_req is ???
liqu: Liquefaction
Attributes
Q_in
Q_out
W
p_req is ???
liqu.inOperation and cooling.inOperation
cooling: CoolingCircuit
Attributes
W
comp: Compression
Attributes
Q_in
Q_out
Secondary is system.uP.secondary, Temp_outMin is ???, Temp_outMax is ???, Q_outMax is ???, p_req is ???
vap.inOperation
comp.inOperation

Appendix D

Diagrams of the graphical method applied to the refined design of the secondary loop of the SMR

This appendix presents the three diagrams for the SMR example discussed in Section 4.2.2 that were produced when applying the new graphical method. The inheritance, internal structure and instance diagrams are presented. The Carnot cycle used as a model is shown Figure D.0.1.

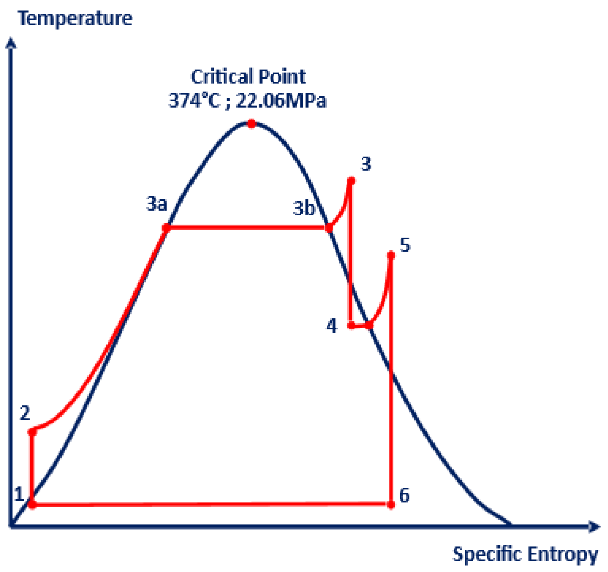
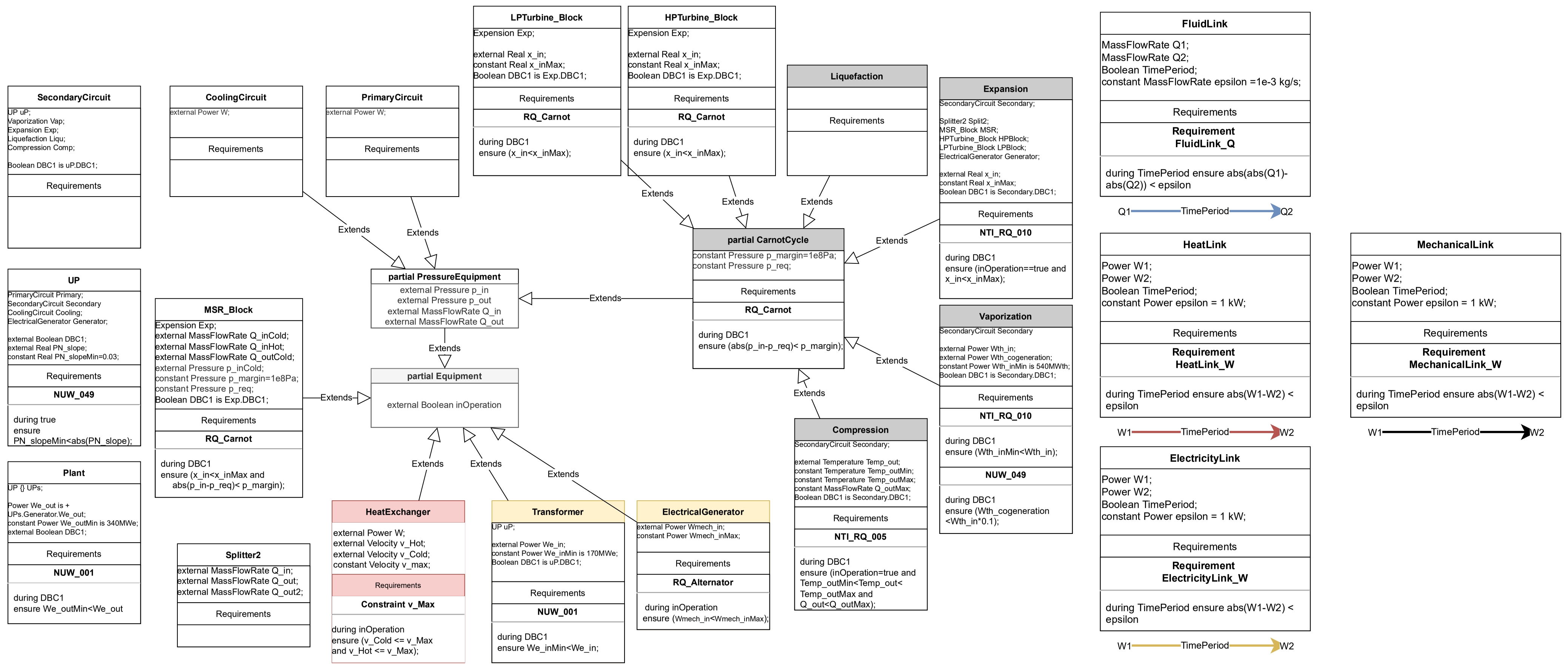
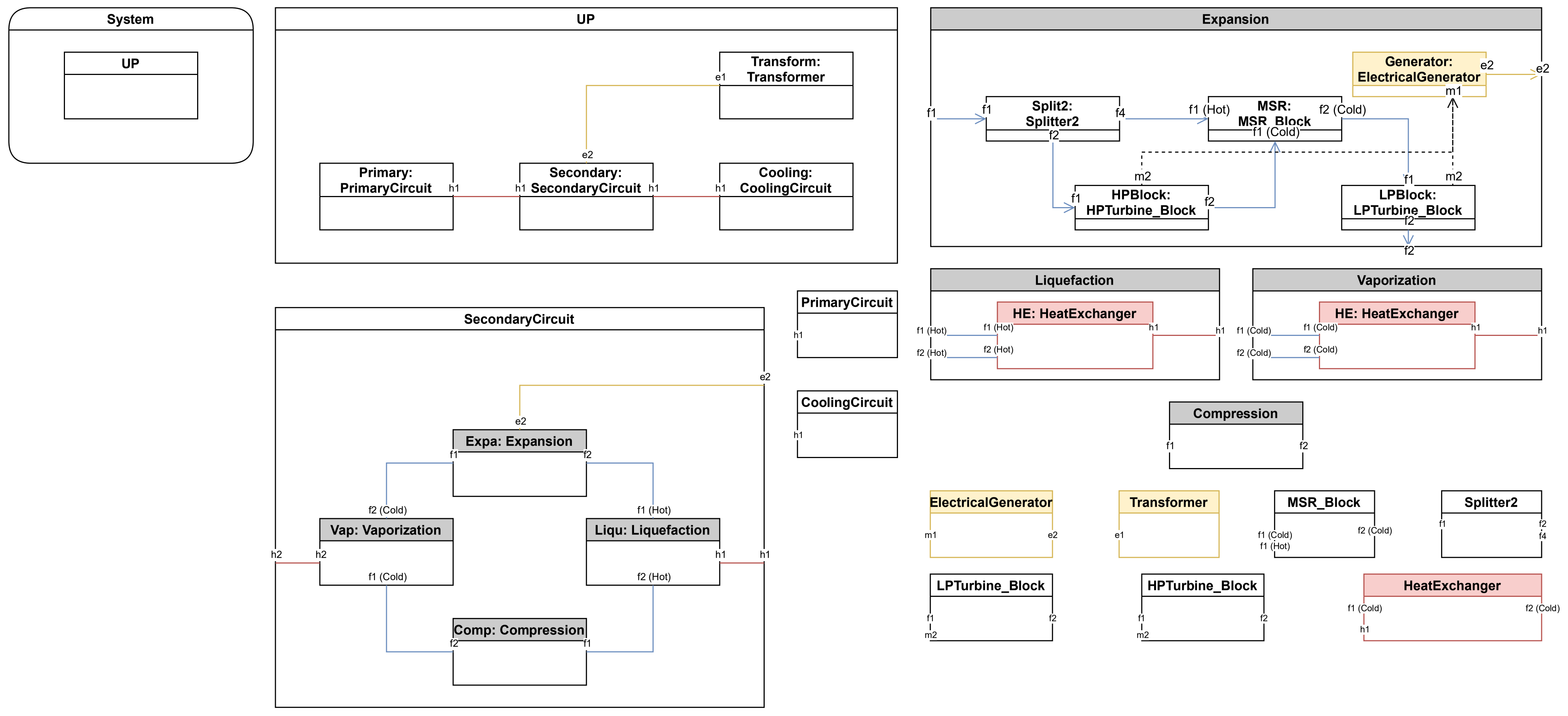
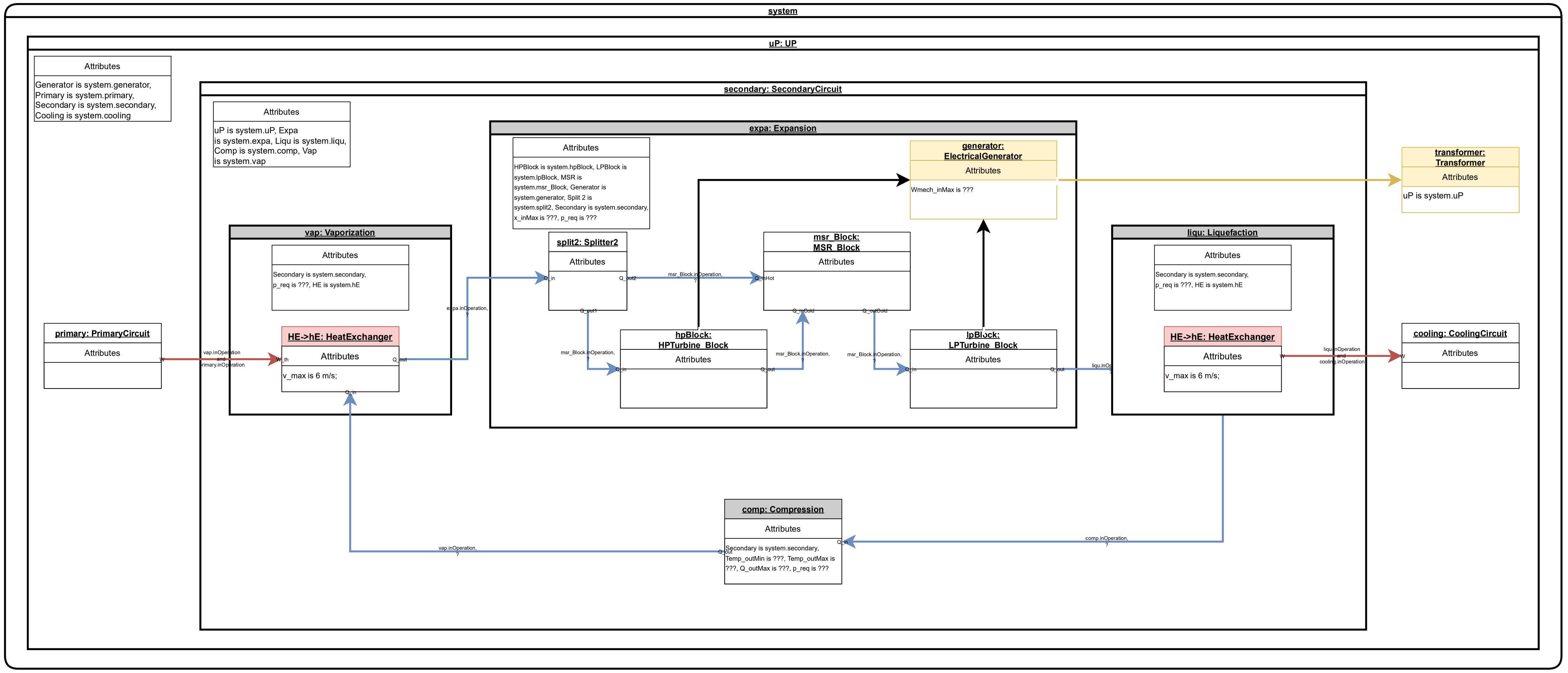


Figure D.0.1: Refined Carnot Cycle for the secondary circuit of a nuclear power plant


SecondaryCircuit
UP uP;
Vaporization Vap;
Expension Exp;
Liquefaction Liqu;
Compression Comp;
Boolean DBC1 is uP.DBC1;
Requirements
CoolingCircuit
external Power W;
Requirements
PrimaryCircuit
external Power W;
Requirements
LPTurbine_Block
Expension Exp;
external Real x_in;
constant Real x_inMax;
Boolean DBC1 is Exp.DBC1;
Requirements
RQ_Carnot
during DBC1
ensure (x_in<x_inMax);
HPTurbine_Block
Expension Exp;
external Real x_in;
constant Real x_inMax;
Boolean DBC1 is Exp.DBC1;
Requirements
RQ_Carnot
during DBC1
ensure (x_in<x_inMax);
Liquefaction
Requirements
Expansion
SecondaryCircuit Secondary;
Splitter2 Split2;
MSR_Block MSR;
HPTurbine_Block HPBlock;
LPTurbine_Block LPBlock;
ElectricalGenerator Generator;
external Real x_in;
constant Real x_inMax;
Boolean DBC1 is Secondary.DBC1;
Requirements
NTI_RQ_010
during DBC1
ensure (inOperation==true and x_in<x_inMax);
FluidLink
MassFlowRate Q1;
MassFlowRate Q2;
Boolean TimePeriod;
constant MassFlowRate epsilon =1e-3 kg/s;
Requirements
Requirement FluidLink_Q
during TimePeriod ensure abs(abs(Q1)-abs(Q2)) < epsilon
Q1
TimePeriod
Q2
Extends
Extends
Extends
Extends
Extends
Extends
UP
PrimaryCircuit Primary;
SecondaryCircuit Secondary
CoolingCircuit Cooling;
ElectricalGenerator Generator;
external Boolean DBC1;
external Real PN_slope;
constant Real PN_slopeMin=0.03;
Requirements
NUW_049
during true
ensure
PN_slopeMin<abs(PN_slope);
partial PressureEquipment
external Pressure p_in
external Pressure p_out
external MassFlowRate Q_in
external MassFlowRate Q_out
Extends
partial CarnotCycle
constant Pressure p_margin=1e8Pa;
constant Pressure p_req;
Requirements
RQ_Carnot
during DBC1
ensure (abs(p_in-p_req)< p_margin);
Vaporization
SecondaryCircuit Secondary
external Power Wth_in;
external Power Wth_cogeneration;
constant Power Wth_inMin is 540MWth;
Boolean DBC1 is Secondary.DBC1;
Requirements
NTI_RQ_010
during DBC1
ensure (Wth_inMin<Wth_in);
NUW_049
during DBC1
ensure (Wth_cogeneration <Wth_in*0.1);
HeatLink
Power W1;
Power W2;
Boolean TimePeriod;
constant Power epsilon = 1 kW;
Requirements
Requirement HeatLink_W
during TimePeriod ensure abs(W1-W2) < epsilon
W1
TimePeriod
W2
MechanicalLink
Power W1;
Power W2;
Boolean TimePeriod;
constant Power epsilon = 1 kW;
Requirements
Requirement MechanicalLink_W
during TimePeriod ensure abs(W1-W2) < epsilon
W1
TimePeriod
W2
MSR_Block
Expension Exp;
external MassFlowRate Q_inCold;
external MassFlowRate Q_inHot;
external MassFlowRate Q_outCold;
external Pressure p_inCold;
constant Pressure p_margin=1e8Pa;
constant Pressure p_req;
Boolean DBC1 is Exp.DBC1;
Requirements
RQ_Carnot
during DBC1
ensure (x_in<x_inMax and abs(p_in-p_req)< p_margin);
Extends
Extends
partial Equipment
external Boolean inOperation
Extends
Extends
Extends
Extends
Extends
Plant
UP {} UPs;
Power We_out is +
UPs.Generator.We_out;
constant Power We_outMin is 340MWe;
external Boolean DBC1;
Requirements
NUW_001
during DBC1
ensure We_outMin<We_out
Splitter2
external MassFlowRate Q_in;
external MassFlowRate Q_out;
external MassFlowRate Q_out2;
Requirements
HeatExchanger
external Power W;
external Velocity v_Hot;
external Velocity v_Cold;
constant Velocity v_max;
Requirements
Constraint v_Max
during inOperation
ensure (v_Cold <= v_Max and v_Hot <= v_Max);
Transformer
UP uP;
external Power We_in;
constant Power We_inMin is 170MWe;
Boolean DBC1 is uP.DBC1;
Requirements
NUW_001
during DBC1
ensure We_inMin<We_in;
ElectricalGenerator
external Power Wmech_in;
constant Power Wmech_inMax;
Requirements
RQ_Alternator
during inOperation
ensure (Wmech_in<Wmech_inMax);
Compression
SecondaryCircuit Secondary;
external Temperature Temp_out;
constant Temperature Temp_outMin;
constant Temperature Temp_outMax;
constant MassFlowRate Q_outMax;
Boolean DBC1 is Secondary.DBC1;
Requirements
NTI_RQ_005
during DBC1
ensure (inOperation=true and Temp_outMin<Temp_out< Temp_outMax and Q_out<Q_outMax);
ElectricityLink
Power W1;
Power W2;
Boolean TimePeriod;
constant Power epsilon = 1 kW;
Requirements
Requirement ElectricityLink_W
during TimePeriod ensure abs(W1-W2) < epsilon
W1
TimePeriod
W2

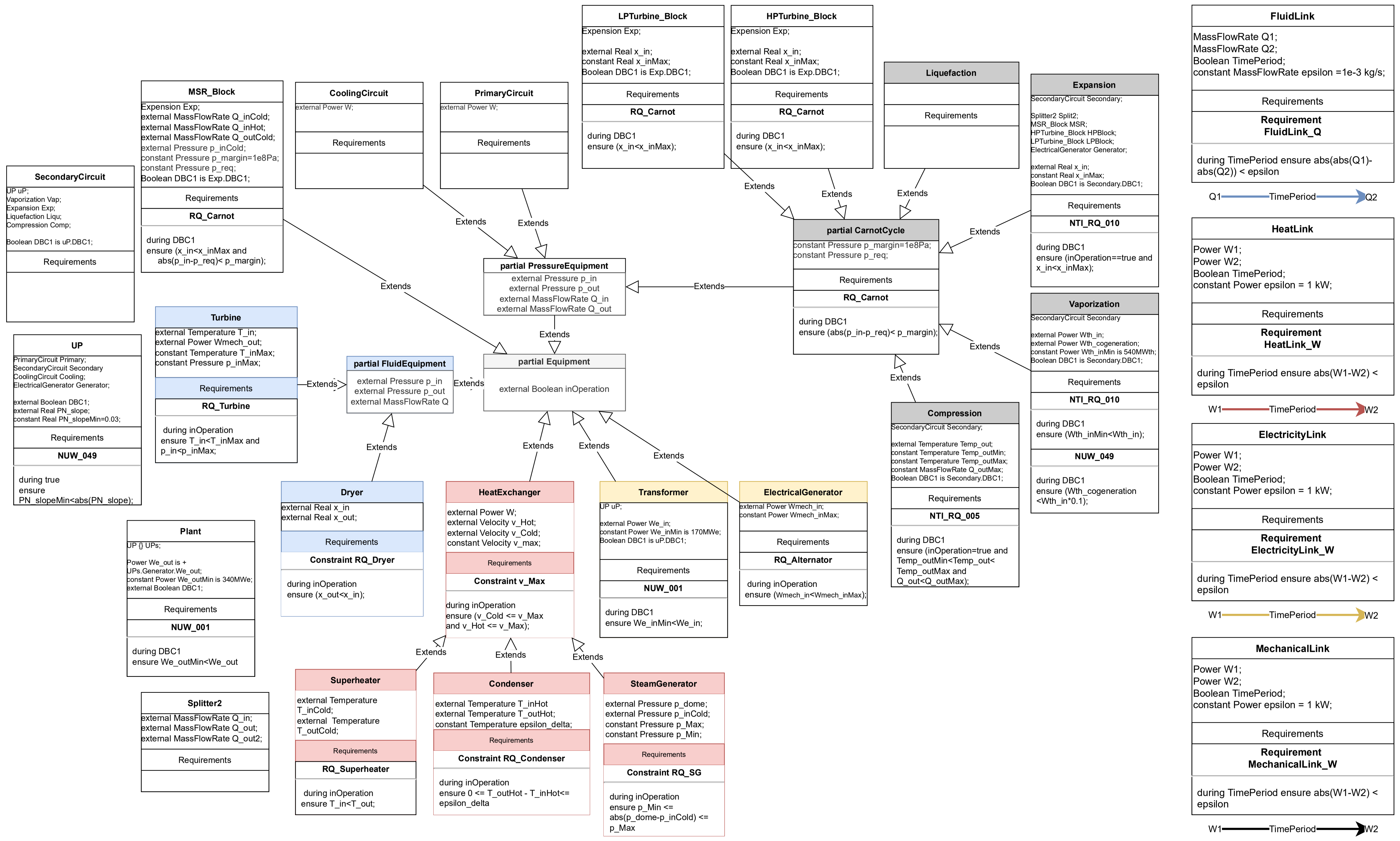

System
UP
UP
Transform: Transformer
e1
e2
Primary: PrimaryCircuit
h1
h1
Secondary: SecondaryCircuit
h1
h1
Cooling: CoolingCircuit
Expansion
f1
f1
Split2: Splitter2
f4
f2
f1 (Hot)
MSR: MSR_Block
f1 (Cold)
f2 (Cold)
Generator: ElectricalGenerator
e2
m1
e2
m2
f1
HPBlock: HPTurbine_Block
f2
f1
m2
LPBlock: LPTurbine_Block
f2
f2
SecondaryCircuit
e2
e2
Expa: Expansion
f1
f2
f2 (Cold)
Vap: Vaporization
h2
h2
f1 (Cold)
f1 (Hot)
Liqu: Liquefaction
h1
h1
f2 (Hot)
Comp: Compression
f2
f1
PrimaryCircuit
h1
CoolingCircuit
h1
Liquefaction
HE: HeatExchanger
f1 (Hot)
f1 (Hot)
f2 (Hot)
f2 (Hot)
h1
h1
Vaporization
HE: HeatExchanger
f1 (Cold)
f1 (Cold)
f2 (Cold)
f2 (Cold)
h1
h1
Compression
f1
f2
ElectricalGenerator
m1
e2
Transformer
e1
MSR_Block
f1 (Cold)
f1 (Hot)
f2 (Cold)
Splitter2
f1
f2
f4
LPTurbine_Block
f1
m2
f2
HPTurbine_Block
f1
m2
f2
HeatExchanger
f1 (Cold)
h1
f2 (Cold)

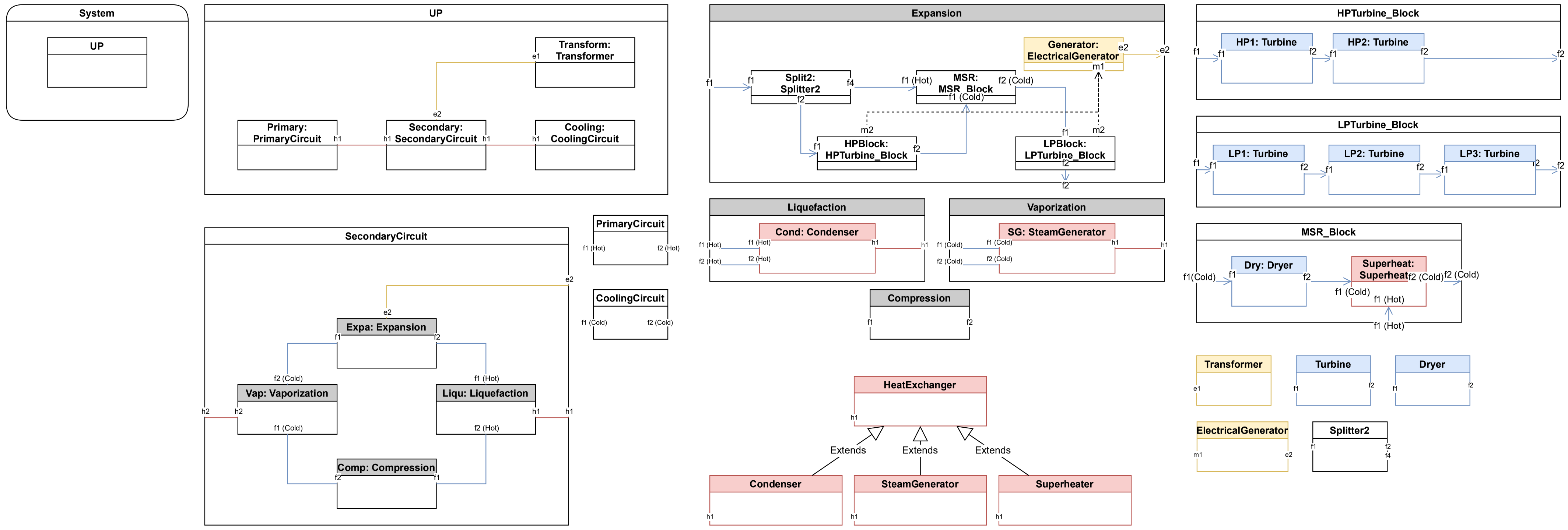

system
uP: UP
Attributes
Generator is system.generator, Primary is system.primary, Secondary is system.secondary, Cooling is system.cooling
primary: PrimaryCircuit
Attributes
W
vap.inOperation and primary.inOperation
secondary: SecondaryCircuit
Attributes
uP is system.uP, Expa is system.expa, Liqu is system.liqu, Comp is system.comp, Vap is system.vap
vap: Vaporization
Attributes
Secondary is system.secondary, p_req is ???, HE is system.hE
HE->hE: HeatExchanger
Attributes
v_max is 6 m/s;
W_th
Q_out
Q_in
expa.inOperation, ?
expa: Expansion
Attributes
HPBlock is system.hpBlock, LPBlock is system.lpBlock, MSR is system.msr_Block, Generator is system.generator, Split 2 is system.split2, Secondary is system.secondary, x_inMax is ???, p_req is ???
split2: Splitter2
Attributes
Q_in
Q_out2
Q_out1
msr_Block.inOperation, ?
msr_Block: MSR_Block
Attributes
Q_inHot
Q_inCold
Q_outCold
hpBlock: HPTurbine_Block
Attributes
Q_in
Q_out
msr_Block.inOperation, ?
msr_Block.inOperation, ?
msr_Block.inOperation, ?
lpBlock: LPTurbine_Block
Attributes
Q_in
Q_out
generator: ElectricalGenerator
Attributes
Wmech_inMax is ???
liqu.inOp
liqu: Liquefaction
Attributes
Secondary is system.secondary, p_req is ???, HE is system.hE
HE->hE: HeatExchanger
Attributes
v_max is 6 m/s;
W
liqu.inOperation and cooling.inOperation
transformer: Transformer
Attributes
uP is system.uP
cooling: CoolingCircuit
Attributes
W
comp: Compression
Attributes
Secondary is system.secondary, Temp_outMin is ???, Temp_outMax is ???, Q_outMax is ???, p_req is ???
Q_in
Q_out
comp.inOperation, ?
vap.inOperation, ?

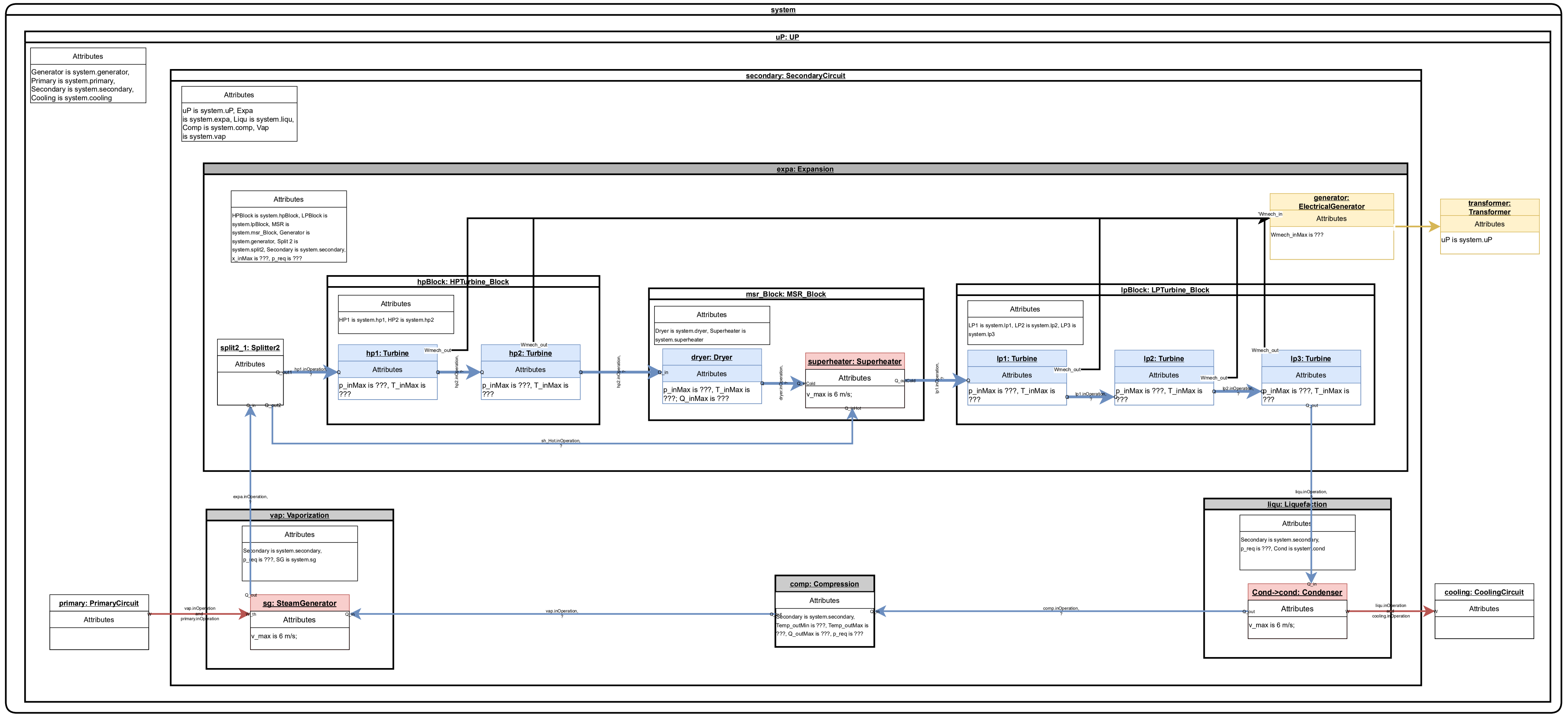
Appendix E

Diagrams of the graphical method applied to the most refined design of the secondary loop of the SMR

This appendix presents the three diagrams for the SMR example discussed in Section 4.2.3 that were produced when applying the new graphical method. The inheritance, internal structure and instance diagrams are presented.


LPTurbine_Block
Expension Exp;
external Real x_in;
constant Real x_inMax;
Boolean DBC1 is Exp.DBC1;
Requirements
RQ_Carnot
during DBC1
ensure (x_in<x_inMax);
HPTurbine_Block
Expension Exp;
external Real x_in;
constant Real x_inMax;
Boolean DBC1 is Exp.DBC1;
Requirements
RQ_Carnot
during DBC1
ensure (x_in<x_inMax);
Liquefaction
Requirements
Expansion
SecondaryCircuit Secondary;
Splitter2 Split2;
MSR_Block MSR;
HPTurbine_Block HPBlock;
LPTurbine_Block LPBlock;
ElectricalGenerator Generator;
external Real x_in;
constant Real x_inMax;
Boolean DBC1 is Secondary.DBC1;
Requirements
NTI_RQ_010
during DBC1
ensure (inOperation==true and
x_in<x_inMax);
FluidLink
MassFlowRate Q1;
MassFlowRate Q2;
Boolean TimePeriod;
constant MassFlowRate epsilon =1e-3 kg/s;
Requirements
Requirement
FluidLink_Q
during TimePeriod ensure abs(abs(Q1)-
abs(Q2)) < epsilon
Q1
TimePeriod
Q2
MSR_Block
Expension Exp;
external MassFlowRate Q_inCold;
external MassFlowRate Q_inHot;
external MassFlowRate Q_outCold;
external Pressure p_inCold;
constant Pressure p_margin=1e8Pa;
constant Pressure p_req;
Boolean DBC1 is Exp.DBC1;
Requirements
RQ_Carnot
during DBC1
ensure (x_in<x_inMax and
abs(p_in-p_req)< p_margin);
CoolingCircuit
external Power W;
Requirements
PrimaryCircuit
external Power W;
Requirements
SecondaryCircuit
UP uP;
Vaporization Vap;
Expansion Exp;
Liquefaction Liqu;
Compression Comp;
Boolean DBC1 is uP.DBC1;
Requirements
Extends
Extends
Extends
Extends
Extends
Extends
Extends
Extends
partial CarnotCycle
constant Pressure p_margin=1e8Pa;
constant Pressure p_req;
Requirements
RQ_Carnot
during DBC1
ensure (abs(p_in-p_req)< p_margin);
partial PressureEquipment
external Pressure p_in
external Pressure p_out
external MassFlowRate Q_in
external MassFlowRate Q_out
Extends
HeatLink
Power W1;
Power W2;
Boolean TimePeriod;
constant Power epsilon = 1 kW;
Requirements
Requirement
HeatLink_W
during TimePeriod ensure abs(W1-W2) <
epsilon
W1
TimePeriod
W2
Vaporization
SecondaryCircuit Secondary
external Power Wth_in;
external Power Wth_cogeneration;
constant Power Wth_inMin is 540MWth;
Boolean DBC1 is Secondary.DBC1;
Requirements
NTI_RQ_010
during DBC1
ensure (Wth_inMin<Wth_in);
NUW_049
during DBC1
ensure (Wth_cogeneration
<Wth_in*0.1);
Extends
Extends
UP
PrimaryCircuit Primary;
SecondaryCircuit Secondary
CoolingCircuit Cooling;
ElectricalGenerator Generator;
external Boolean DBC1;
external Real PN_slope;
constant Real PN_slopeMin=0.03;
Requirements
NUW_049
during true
ensure
PN_slopeMin<abs(PN_slope);
Turbine
external Temperature T_in;
external Power Wmech_out;
constant Temperature T_inMax;
constant Pressure p_inMax;
Requirements
RQ_Turbine
during inOperation
ensure T_in<T_inMax and
p_in<p_inMax;
Extends
partial FluidEquipment
external Pressure p_in
external Pressure p_out
external MassFlowRate Q
Extends
partial Equipment
external Boolean inOperation
Extends
Compression
SecondaryCircuit Secondary;
external Temperature Temp_out;
constant Temperature Temp_outMin;
constant Temperature Temp_outMax;
constant MassFlowRate Q_outMax;
Boolean DBC1 is Secondary.DBC1;
Requirements
NTI_RQ_005
during DBC1
ensure (inOperation=true and
Temp_outMin<Temp_out<
Temp_outMax and
Q_out<Q_outMax);
ElectricityLink
Power W1;
Power W2;
Boolean TimePeriod;
constant Power epsilon = 1 kW;
Requirements
Requirement
ElectricityLink_W
during TimePeriod ensure abs(W1-W2) <
epsilon
W1
TimePeriod
W2
Extends
Extends
Extends
Extends
Dryer
external Real x_in
external Real x_out;
Requirements
Constraint RQ_Dryer
during inOperation
ensure (x_out<x_in);
HeatExchanger
external Power W;
external Velocity v_Hot;
external Velocity v_Cold;
constant Velocity v_max;
Requirements
Constraint v_Max
during inOperation
ensure (v_Cold <= v_Max
and v_Hot <= v_Max);
Transformer
UP uP;
external Power We_in;
constant Power We_inMin is 170MWe;
Boolean DBC1 is uP.DBC1;
Requirements
NUW_001
during DBC1
ensure We_inMin<We_in;
ElectricalGenerator
external Power Wmech_in;
constant Power Wmech_inMax;
Requirements
RQ_Alternator
during inOperation
ensure (Wmech_in<Wmech_inMax);
Plant
UP {} UPs;
Power We_out is +
UPs.Generator.We_out;
constant Power We_outMin is 340MWe;
external Boolean DBC1;
Requirements
NUW_001
during DBC1
ensure We_outMin<We_out
MechanicalLink
Power W1;
Power W2;
Boolean TimePeriod;
constant Power epsilon = 1 kW;
Requirements
Requirement
MechanicalLink_W
during TimePeriod ensure abs(W1-W2) <
epsilon
W1
TimePeriod
W2
Extends
Extends
Extends
Splitter2
external MassFlowRate Q_in;
external MassFlowRate Q_out;
external MassFlowRate Q_out2;
Requirements
Superheater
external Temperature
T_inCold;
external Temperature
T_outCold;
Requirements
RQ_Superheater
during inOperation
ensure T_in<T_out;
Condenser
external Temperature T_inHot
external Temperature T_outHot;
constant Temperature epsilon_delta;
Requirements
Constraint RQ_Condenser
during inOperation
ensure 0 <= T_outHot - T_inHot<=
epsilon_delta
SteamGenerator
external Pressure p_dome;
external Pressure p_inCold;
constant Pressure p_Max;
constant Pressure p_Min;
Requirements
Constraint RQ_SG
during inOperation
ensure p_Min <=
abs(p_dome-p_inCold) <=
p_Max


System
UP
Transform: Transformer
Primary: PrimaryCircuit
Secondary: SecondaryCircuit
Cooling: CoolingCircuit
SecondaryCircuit
Expa: Expansion
Vap: Vaporization
Liqu: Liquefaction
Comp: Compression
PrimaryCircuit
CoolingCircuit
Expansion
Split2: Splitter2
MSR: MSR_Block
Generator: ElectricalGenerator
HPBlock: HPTurbine_Block
LPBlock: LPTurbine_Block
Liquefaction
Cond: Condenser
Vaporization
SG: SteamGenerator
Compression
HeatExchanger
Extends
Condenser
SteamGenerator
Superheater
HPTurbine_Block
HP1: Turbine
HP2: Turbine
LPTurbine_Block
LP1: Turbine
LP2: Turbine
LP3: Turbine
MSR_Block
Dry: Dryer
Superheat: Superheater
Transformer
Turbine
Dryer
ElectricalGenerator
Splitter2
f1 (Hot)
f2 (Hot)
f1 (Cold)
f2 (Cold)
e1
e2
h1
h2
m1
m2
f1
f2
f4


system
uP: UP
Attributes
Generator is system.generator, Primary is system.primary, Secondary is system.secondary, Cooling is system.cooling
secondary: SecondaryCircuit
Attributes
uP is system.uP, Expa is system.expa, Liqu is system.liqu, Comp is system.comp, Vap is system.vap
expa: Expansion
Attributes
HPBlock is system.hpBlock, LPBlock is system.lpBlock, MSR is system.msr_Block, Generator is system.generator, Split 2 is system.split2, Secondary is system.secondary, x_inMax is ???, p_req is ???
split2_1: Splitter2
Attributes
Q_out1
Q_in
Q_out2
hp1.inOperation, ?
hpBlock: HPTurbine_Block
Attributes
HP1 is system.hp1, HP2 is system.hp2
hp1: Turbine
Attributes
p_inMax is ???, T_inMax is ???
Wmech_out
hp2.inOperation, ?
hp2: Turbine
Attributes
p_inMax is ???, T_inMax is ???
Wmech_out
hp2.inOperation, ?
msr_Block: MSR_Block
Attributes
Dryer is system.dryer, Superheater is system.superheater
dryer: Dryer
Attributes
p_inMax is ???, T_inMax is ???; Q_inMax is ???
Q_in
dryer.inOperation, ?
superheater: Superheater
Attributes
v_max is 6 m/s;
Q_inCold
Q_outCold
Q_inHot
sh_Hot.inOperation, ?
lp1.inOperation, ?
lpBlock: LPTurbine_Block
Attributes
LP1 is system.lp1, LP2 is system.lp2, LP3 is system.lp3
lp1: Turbine
Attributes
p_inMax is ???, T_inMax is ???
Wmech_out
lp1.inOperation, ?
lp2: Turbine
Attributes
p_inMax is ???, T_inMax is ???
Wmech_out
lp2.inOperation, ?
lp3: Turbine
Attributes
p_inMax is ???, T_inMax is ???
Wmech_out
Q_out
generator: ElectricalGenerator
Attributes
Wmech_inMax is ???
Wmech_in
transformer: Transformer
Attributes
uP is system.uP
expa.inOperation, ?
vap: Vaporization
Attributes
Secondary is system.secondary, p_req is ???, SG is system.sg
sg: SteamGenerator
Attributes
v_max is 6 m/s;
Q_out
W_th
Q_in
primary: PrimaryCircuit
Attributes
W
vap.inOperation and primary.inOperation
vap.inOperation, ?
comp: Compression
Attributes
Secondary is system.secondary, Temp_outMin is ???, Temp_outMax is ???, Q_outMax is ???, p_req is ???
Q_out
Q_in
comp.inOperation, ?
liqu.inOperation, ?
liqu: Liquefaction
Attributes
Secondary is system.secondary, p_req is ???, Cond is system.cond
Cond->cond: Condenser
Attributes
v_max is 6 m/s;
Q_in
Q_out
W
liqu.inOperation and cooling.inOperation
cooling: CoolingCircuit
Attributes
W