



Herausragende Masterarbeiten

Autor*in

Studiengang

Masterarbeitstitel

R
TU
P

Distance and Independent
Studies Center
DISC

Technische Universität Kaiserslautern
Distance Study Program
Software Engineering for Embedded Systems

Master's Thesis

Case study:
SysMD in industrial application

Provided by

Nicolai Birkemose Nielsen, Master thesis student
Bachelor of Engineering, Electrotechnics

First supervisor: Prof. Christoph Grimm

Second supervisor: Mr. Martin Møller Steffensen

Declaration

Please leave the text below in German!!! It means: "I certify that I prepared this master thesis independently, using only the sources and aids stated, and that I have indicated as such the parts of the sources used literally or in substance."

Ich versichere, dass ich diese Masterarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Place, date

Signature

Sønderborg, Denmark, 2023-01-26

Abstract

The aim of this thesis is to perform a case study to investigate the usability of SysMD in industrial applications. The focus is on how well it can bridge the gap between requirement specifications, modeling, and actual development.

SysMD is a new documentation and modeling language which aims to bring documentation and modeling closer together while still not requiring the user to be an expert in modeling or requirement specification. This differentiates SysMD from other tools which focus on either documentation, modeling, or are aimed at modeling experts.

This thesis will show through the case study part that SysMD as a language has a good future with potential of being used as a language bridging the gap between requirements, documentation, and modeling without the user needing to be an expert within modeling. It will also show that SysMD Notebook in its current state is not ready for primetime, and I give recommendations on how to improve both the SysMD language as well as the SysMD Notebook to make it usable for industrial projects in the future.

Table of Contents

1	Introduction	2
1.1	Project Justification	2
1.2	Objectives	4
1.3	Methodology.....	4
1.4	Scope	5
1.5	Limitations.....	5
2	State of the art	6
2.1	Background.....	6
2.2	Document based approach.....	7
2.2.1	DOORS	7
2.2.2	Markdown	8
2.2.3	Excel.....	10
2.2.4	Polarion	11
2.3	Model based approach /mixed approach	12
2.3.1	Enterprise Architect.....	12
2.4	Notebook approach (combined approach).....	13
2.4.1	Jupyter Notebook	13
2.4.2	SysML v2	14
2.5	Modeling and requirement tools – Comparison	15
2.5.1	Comparison	15
2.5.2	Criteria.....	17
2.6	SysMD tool – Deep dive	20
2.6.1	SysMD language	20
2.6.2	SysMD Recommender System.....	23
2.6.3	SysMD Notebook.....	24
3	SysMD in industrial application.....	27
3.1	Project description for case study.....	27
3.1.1	Focus	28
3.1.2	ECL product family.....	29
3.2	Project in SysMD.....	31

3.2.1	Project structure.....	31
3.2.2	Product overview.....	32
3.2.3	SysMD templates	36
3.2.4	Modules.....	37
3.2.5	Analysis with SAT/SMT solver in relation to consistency.....	39
3.2.6	Computation of requirements.....	40
3.2.7	hasA relationship tree-view.....	42
4	Results.....	43
4.1	Analysis with SAT/SMT solver	43
4.2	Ability to link documentation with models.....	44
4.2.1	Refine documentation with added models	44
4.3	Quality of SysMD Notebook.....	45
4.4	Improvements in SysMD.....	46
4.4.1	Improved error information.....	46
4.4.2	Improved relationship syntax	47
4.4.3	Diagrams support	49
4.4.4	General improvements of SysMD Notebook.....	50
4.4.5	MD file linking support.....	50
4.4.6	Table of content support.....	51
4.4.7	Version control support.....	51
4.5	Validity.....	52
4.6	Usability.....	53
4.6.1	SysMD language	53
4.6.2	SysMD Notebook.....	54
5	Conclusion.....	55
	References	57
	Appendix	61
	Appendix: A SysMD project files	61
	Appendix: B Tools comparison evaluation	77

Abbreviations

API	Application Programming Interface
CRS	Customer requirement specification
DHW	Domestic hot water. This refers to the warm tap water in households.
DOORS	Dynamic Object Oriented Requirements System
Doxygen	Language and generator for generating documentation for code source.
EA	Enterprise Architect
ECL	Electronic Controller from Danfoss for controlling district heating applications.
Eclipse	Open-source Integrated Development Environment
IO	Input/Output. This refers to the physical inputs and outputs of the hardware.
ITE	Function in SysMD – “If ... Then ... Else ...”
md files	Markdown files
PRS	Product requirement specification
SAT	Boolean satisfiability problem
SMT	Satisfiability modulo theories
SysMD	Textual modeling language inspired by SysML v2 + Markdown(MD)
SysML v2	System Modeling Language – Release version 2
UML	Unified Modeling Language
VBA	Visual Basic for Application
VS Code	Visual Studio Code

List of Tables

Table 1 - Comparison table between different requirement tools.....	15
Table 2 - IOs comparison table of ECLs.....	30
Table 3 - Comparison between correct Modbus T1.5 and T3.5 values and calculated.	43
Table 4 - Statistics for issue tracking of SysMD in GitLab for the last ~6 months.	45

List of Figures

Figure 1 - Overview of district heating [1].	2
Figure 2 - Example of flat station application with radiator and domestic hot water.	3
Figure 3 - Visual view of Markdown files in ECL software.....	9
Figure 4 – Small extract of md file containing design specification for Inter Device Sync component. Shown in both raw format (left side) and visualized view (right side).	9
Figure 5 - (Redacted) Real-world example of PRS in Excel [10].....	10
Figure 6 - Agila Recommender System in action [25].....	23
Figure 7 - Example of SysMD document in Visual Studio Code and SysMD Notebook.	24
Figure 8 - Constraint check example.....	25
Figure 9 - Example of isA and hasA tree-views in SysMD Notebook.....	26
Figure 10 - Overview of ECL x20 Platform.	27
Figure 11 - Visualization of ECL 220 and ECL 120.....	29
Figure 12 - Example of flat station application with radiator and domestic hot water.	29
Figure 13 - Project structure in SysMD Notebook.....	31
Figure 14 - Document imports in Product Overview.....	32
Figure 15 - Product and variant definition.	32
Figure 16 - Specifying list of available types of IOs.....	33
Figure 17 - ECL product family specification.	34
Figure 18 - ECL 120 IO specification in SysMD.....	35
Figure 19 - Start of TemplateModule.md file.	36
Figure 20 - Start of temperature sensor module.....	37
Figure 21 – Example of some requirements from temperature sensor module.	38
Figure 22- Example of some requirements from RS485 module.....	38
Figure 23 - Example of function specification in temperature sensor module.	39
Figure 24 - Example of inconsistency check.	40
Figure 25 - T1.5 and T3.5 timeouts implementation in SysMD with calculated results.	41
Figure 26 - hasA relationship tree-view of package Product	42
Figure 27 – Example of bad error information.	46
Figure 28 - Component diagram of ECL product overview	49

Acknowledgement

I would like to thank my university supervisors, Professor Christoph Grimm, and Sandor Dalecke, for their continuous support throughout this project. Without their support it would not have been possible to successfully finish this thesis.

A big thanks to Danfoss and specially to my boss and company supervisor, Martin Møller Steffensen, for supporting me through the last 2.5 years while I have been taking this master besides work.

Thanks also to my new boss, Jens Christian Bentzon, and my colleagues for their support and understanding through this last half year where I have been less available at work while I have been writing my thesis.

Thanks of course to my girlfriend and family for their support and understanding during these last 2.5 years.

1 Introduction

1.1 Project Justification

District heating is common today in many cities for supplying heating to commercial, industrial, and residential buildings of the city. These networks contain several electronic components in various places of the district heating network.

- Sub stations and valves in the network
- Flat stations at end customers – Example of such application can be seen in Figure 2.

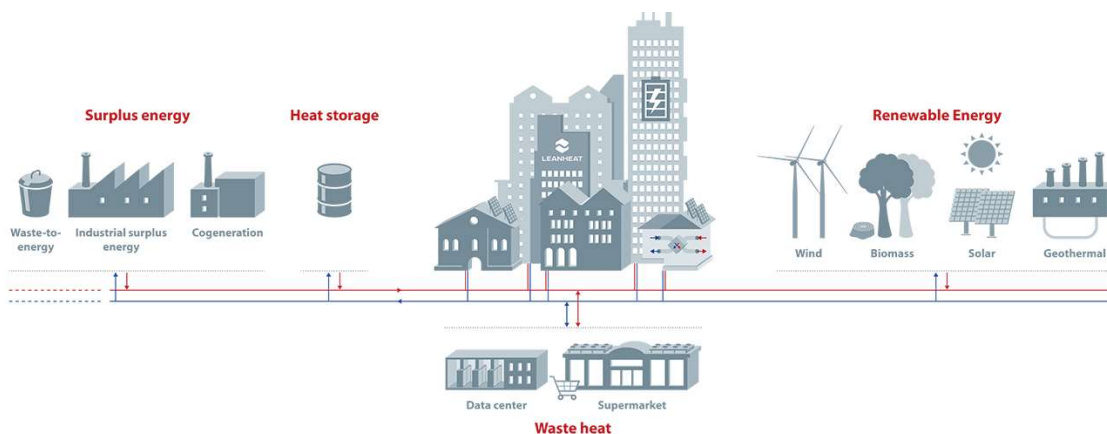


Figure 1 - Overview of district heating [1].

Many of the components could take advantage from being able to communicate with each other to both improve efficiency and comfort of the heating [1]. This specially applies for the sub- and flat stations which can consist of a mix of electronic components:

- Intelligent Controller – Proactively controls the supplied heat based on actual need and weather condition.
- Differential pressure control – Ensuring correct pressure through the system.
- Actuators – Controlling valves to adjust amount of supplied heat.
- Intelligent actuators – Actuators that independently can control valves to optimize and balance network based on reference settings from e.g., a Building Management System.
- Heat meter – Used for measuring amount of energy (heat) used and billing the customer for usage.
- Pumps – Ensure proper flow of water in the system.
- Sensors – Different sensors necessary for enabling proper control by the controller. This can be e.g., temperature, flow, and differential pressure sensors.

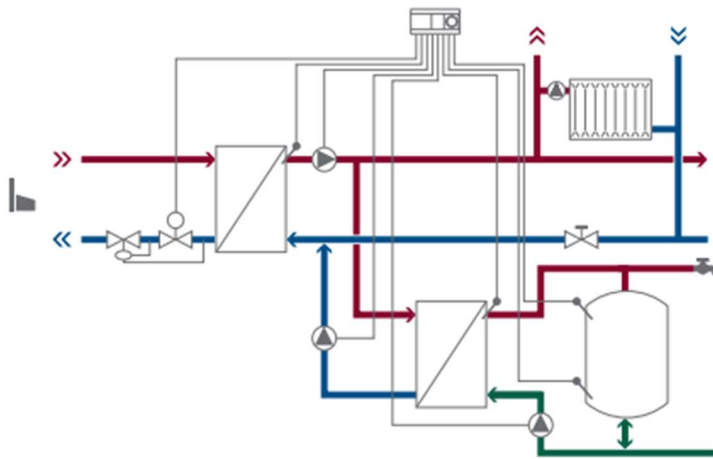


Figure 2 - Example of flat station application with radiator and domestic hot water.

Today these components are often developed separately from each other and with each having a long product lifespan, that makes it difficult to ensure compatibility across all products. Requirements for these components are often not directly aligned between them.

For enabling these products to communicate with each other, it is important that requirements about products interaction are well aligned across products [2]. This is easier ensured by having easy access to useful documentation and requirements tools, which is where SysMD comes into play. SysMD is planned to be released as an open-source tool which makes it freely accessible. Besides that, it claims it will be easier to use by everyone without long tool introduction. This is primarily because it has a simple syntax with a combination of models in SysMD language and documentation in Markdown. The SysMD language is compatible with SysML v2 to ensure interoperability. This mix should make it easily readable without prior knowledge of programming languages.

This case study will check the usability of SysMD in real world environment and compare it to current state of the art documentation and requirements tools.

1.2 Objectives

The objectives of this case study are to check the usability of SysMD in industrial applications. This takes reference in a real product line from Danfoss with primary usage in district heating.

Main objectives consist of:

- Investigate and compare functionality of SysMD with various state of the art commercial tools.

This includes but is not limited to following:

- Qualify requirement tracking across products.
- Qualify requirement tracking on application interfaces level.
- Investigate possibility of auto-generated architectural diagrams.
- Qualify requirement traceability to code.
- Versioning of requirement documentation.

- Document requirements of the ECL in SysMD as a usability proof of concept.

This includes but is not limited to following:

- Analysis with SAT/SMT solver in relation to consistency.
- Ability to link documentation with models.
- Refine documentation with added models.

1.3 Methodology

The *State of the art* research phase is performed in 2 steps:

- General search and analysis of various tools within the field of requirement management. This part does not contain any real usage of the specific tools and is purely an analytic exercise.
- Authors personal experience with different requirement tools.

The *SysMD in industrial application* case study part of this thesis is performed as a full practical experience. The practical experience consists of doing a thorough software documentation of ECL which is a product in a product line from Danfoss. This product already has a mixed level of documentation spread across documents in Word, Excel, and Polarion. Therefore, the main task is to take this documentation and transform it into SysMD and create any extra missing

information. There are rules around selecting which part of the product to document, which are to ensure this covers a broad range of functionality and use cases which a requirement management tool should cover.

The knowledge gained from the case study is then used to compare SysMD against the tools from *State of the art* chapter.

1.4 Scope

Scope for *State of the art* section is limited to a few tools within each group of tools as the focus is on testing the capabilities of SysMD and comparing them to existing tools. All tools analyzed are focused on their capabilities in requirement engineering area even though they might be greatly capable in other areas.

The scope of this thesis is limited to the relevant documentation for the ECL, which means the documentation added in SysMD for the case study will not be artificially created for the purpose of testing SysMD but will have ongoing relevant usage in the actual product.

1.5 Limitations

The ECL is a product from Danfoss. This means that parts of documentation created in SysMD for the ECL product is considered confidential and cannot be directly disclosed to the public without being redacted. Parts of documentation in this case study have therefore been either redacted or “faked” to avoid disclosing proprietary information. All values for requirements in this report does not represent the real requirements of the Danfoss products. SysMD is under development and can therefore lack features and be unstable.

2 State of the art

This chapter will go in depth about a few different “state of the art” documentation and requirement tools and how they stack up against SysMD. The definition of “state of the art” in this context is not necessarily about this is the latest or most advanced, but more about what is widely used and known [3] [4].

Section *2.5 Modeling and requirement tools – Comparison* contains a comparison of the different tools described in the following. The comparison is based on both a technical assessment but also includes my personal experience and opinion.

2.1 Background

There are many different documentation and requirement tools, and many of them follows some similar approaches for how to use them. In this section these tools have been grouped into three different types based on their overall approach type:

- Document based
- Model based / Mixed based
- Notebook based

The description for each tool in the following sections is kept on a technical level avoiding any non-technical comparisons. Any comparisons are kept for section *2.5 Modeling and requirement tools – Comparison* to avoid any confusion between what are “personal preferences” and clear facts.

2.2 Document based approach

Document based approach refer to type of approach where you, in general, have a single document. It is also possible to have several documents however it is less common. These documents contain only static information (text, pictures, graphs), there is no code or other dynamically calculated parts.

2.2.1 DOORS

Rational Dynamic Object Oriented Requirements System (DOORS) is full solution system for requirement management developed by IBM. DOORS is considered one of the leading requirement management systems available [4], it currently comes in two versions DOORS and DOORS Next. Both systems are highly capable systems within the requirement management world and supports [5] [6] [7]:

- Web browser for access to requirement database.
- (Only DOORS) PC client for offline/online manage of requirements.
- Full traceability from CRS to final product including design items, test plans, etc.
- Change management process including review, ensuring no change to requirements becomes “publicly” available before they have been reviewed and approved.
- Integration with many third party tools.
- Various attributes.
- Strong filtering tool.
- (Only DOORS Next) Team collaboration with dashboards, reviews, and comments.
- (Only DOORS Next) AI to help improve quality of requirements.

The tools do not utilize SAT/SMT for validating that there are no conflicting requirements and instead fully relies on their review process to ensure all requirements is valid.

While DOORS can be used both in offline and online mode, DOORS Next is purely online.

2.2.2 Markdown

Markdown is a lightweight markup language to be used for creating formatted text. In many ways it works with any plain-text editors and can easily be used by anyone. The plain text can be visualized with headings, pictures, and links as you would in any Word document. This is achieved by simple text syntax.

When referring to Markdown it does not simply mean one single file format that is interpreted in the same way by all tools. Markdown is an open file format and in the beginning it was not fully standardized and it contained ambiguities and unanswered questions about how it should be implemented which caused it to be used as inspiration for a bunch of variations which all added their own extra features on top of the original Markdown. In 2014, approximately 10 years after Markdown was created, CommonMark was released in its first version which is now considered the standardized version of Markdown [8].

The simplicity of how Markdown works, and its initial user purpose are reasons why Markdown itself lacks a lot of features which are common for many requirements specification tools. This makes Markdown as a standalone tool less suitable for use in requirement specification. This simplicity of Markdown makes it a good candidate for building extensions on top, to handle features related to requirement specification and traceability. This can already be seen today as many tools for various user purposes have Markdown for their editor [8], and many known sites and projects within the software world have adopted Markdown [9].

The ECL project which is in the scope of this case study is already using Markdown across parts of the software. A visual view of Markdown usages in ECL project can be seen in Figure 3, the red marked text visualizes where md files are used across the software:

- Changelog for the entire software.
- Protocol specification, which is also shared with externals as auto-generated PDF.
- Design specification for individual components.
- Doxygen generated output of entire software includes the design specification md files from individual software modules.

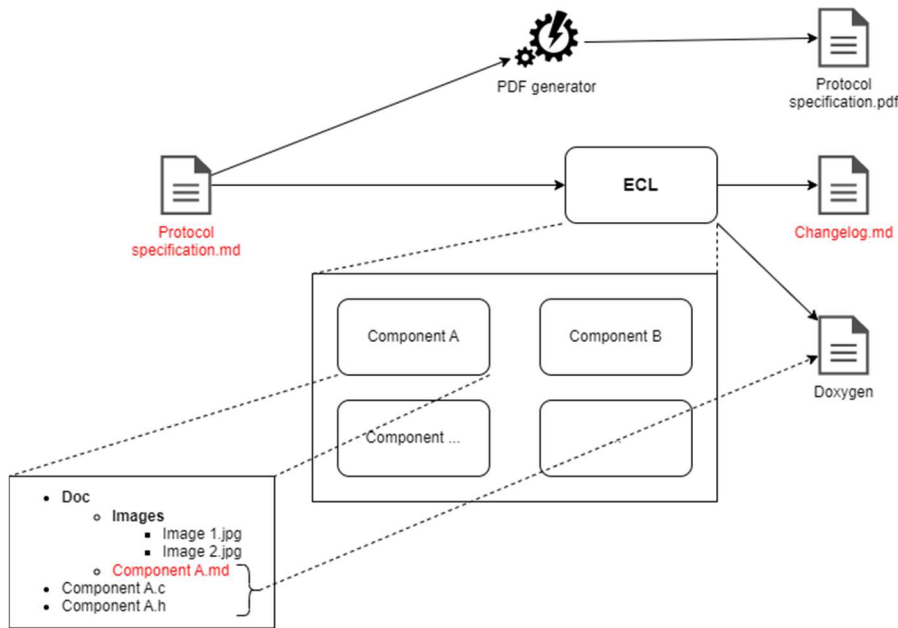


Figure 3 - Visual view of Markdown files in ECL software.

Example snippet of md file in a specific component, in this case Inter Device Sync, can be seen below. This shows a section of the raw md file and how that is visually rendered.

```

## Process Flow

The "Inter Device Synchronization" module executes within the
"SystemControl" and "NCP_Communication" scopes. From the overall
architecture drawing the following can be observed regarding the
module's execution flow;
- Upon system initialization, "SystemControl" initializes the module
and its sub-components. On top of this, registration for a callback
upon reception of the synchronization message is also done. See
sequence diagram ("Process Flow [1] - System Initialization" below).

!![Overview](Images/ProcessFlow1.png)

- In case there is need to send a synchronization signal request to
remote devices, e.g. at boot-up it is done by "SystemControl", a
request to send it can be done. See sequence diagram ("Process Flow [2]
- Sending a Synchronization Signal Request to Remote Devices" below)
where a message is encoded and sent to a remote.

!![Overview](Images/ProcessFlow2.png)

Upon reception of a synchronization signal request from a remote
device, a response is encoded and sent back. Synchronization action is
later done. See sequence diagram ("Process Flow [3] - Receiving a
Synchronization Signal Request from a Remote Device" below).

!![Overview](Images/ProcessFlow3.png)

Upon reception of a synchronization signal response from a remote
device, a flag indicating to send synchronization request is reset. See
sequence diagram ("Process Flow [4] - Receiving a Synchronization
Signal Response from a Remote Device" below).

!![Overview](Images/ProcessFlow4.png)
                
```

Process Flow

The "Inter Device Synchronization" module executes within the "SystemControl" and "NCP_Communication" scopes. From the overall architecture drawing the following can be observed regarding the module's execution flow;

- Upon system initialization, "SystemControl" initializes the module and its sub-components. On top of this, registration for a callback upon reception of the synchronization message is also done. See sequence diagram ("Process Flow [1] - System Initialization" below).

- In case there is need to send a synchronization signal request to remote devices, e.g. at boot-up it is done by "SystemControl", a request to send it can be done. See sequence diagram ("Process Flow [2] - Sending a Synchronization Signal Request to Remote Devices" below) where a message is encoded and sent to a remote.

Figure 4 – Small extract of md file containing design specification for Inter Device Sync component. Shown in both raw format (left side) and visualized view (right side).

2.2.3 Excel

Excel is a well-known program by Microsoft and many people are familiar with it. Excel is a versatile tool that can be used for a lot of things including requirement specifications. While Excel is a very strong tool it lacks one important key aspect which is traceability tracking. This means that any traceability tracking of requirements, e.g., customer requirements to product requirements will require manual tracking from user side.

Excel provides a lot of functionality needed in a good requirement tool: good overview, filtering, ease of use, categorization, and review. Below shows a real-world example of product requirement specification (PRS) in Excel.

	A	B	C	D	E	F	G	H	I	J	L	M	
	ID	Product A	Product B	Product C	Q	M	S	C	Topic	Sub topic	Requirement	Resp	Comments
2									Enclosure				
3	PR-100		X	X					Enclosure	Size of enclosure	IEC 61554 standard: W x H x D: 144 x 96 x 62,5 (144 x 96 x 62,5)	Person A	
4	PR-101	X		X					Enclosure	Size of enclosure	W x H x D lower than: 390 x 120 x 130 (276x100x60)	Person A	
5	PR-102	X	X	X					Enclosure	User interface	5 LED s. One button; 2 press each day - class 6.10 EN60730-1.	Person B	
6	PR-103	X	X	X					Enclosure	Product labels	Label with unique serial #, electrical ratings and approval markings.	Person B	
7	PR-104	X	X	X					Enclosure	terminals	All terminals must be numbered/labeled	Person A	
8	PR-105	X	X	X					Enclosure	Mounting	DIN Rail mounting - Symmetrical DIN 3 (35 mm)	Person C	
9	PR-106	X	X	X					Enclosure	Mounting	Wall Mounting		
10	PR-107		X	X					Enclosure	Mounting	Panel Mounting cut-out size: W 138 [mm], H 92 [mm]; IEC 61554, panel thickness: 5 [mm]		
11	PR-108	X			IP 40		IP 41	IP 54	Enclosure	Ingress protection	IEC 60529 Wall mounting, cable positioned downwards, with exception of power cable		
12	PR-109		X		IP 40		IP 41		Enclosure	Ingress protection	IEC 60529 Panel mounting, back plate min. IP-20 Cable positioned downwards		
13									Environment				
14	PR-201	X	X	X					Lifecycle		Minimum 10 years lifetime (component selection Hardware)		
15	PR-202	X	X	X					Operation Ambient Temperature		IEC 60721-3-3 3K4 /3K22 5°C to 45°C	Person D	
16	PR-203	X	X		X				Operation Ambient Temperature		IEC 60721-3-3 3K5 /3K23 -5°C to 55°C	Person D	

Figure 5 - (Redacted) Real-world example of PRS in Excel [10].

The example shows how Excel can easily support:

- Unique requirement IDs.
- Product specific details.
- MoSCoW method [11] (in this case modified version with Q as qualifier).
- Requirement grouping (Topic, Subtopic).
- Assigned owner of requirement.

Requirement IDs and ID linking is something that needs to be performed manually without any tool support. It results in an increased risk that part of the requirement specification is being flawed.

While it is stated here that requirement IDs must be handled manually, it is possible to handle this in an automatic way using custom VBA macros inside Excel. This will require custom VBA macros which must be handled and maintained by the document owner.

As many other tools, this does not have any SAT/SMT checking and relies fully on the users to ensure there are no conflicting requirements and that requirements are valid.

2.2.4 Polarion

Polarion is a full solution system by Siemens Industry Software Inc. It provides a full solution for the software lifecycle, from specification to release including planning, coding, testing, and management.

In this context the focus will be on the document part of Polarion. Documents in Polarion can be structured in same way as it would be structured in any other documentation tool e.g. like Microsoft Word. Additionally to working as a normal document, it can also mark any part of the document as a requirement which can be tracked through the entire software lifecycle.

When using Polarion through the entire software lifecycle, the tool can be used to track any single requirement and its impact in the project. A requirement can be traced in both directions, from customer requirement to development and testing, or the other way from a specific test to which customer requirements are covered by that specific test. The tool keeps a lot of meta data to every single requirement to ensure traceability, this includes comments, approvals, linked requirements, status, governance, restriction, functional safety, ... and many more [12].

The tool does not have any SAT/SMT checking and relies fully on their review and approval flows to ensure there is no conflicting requirements and that requirements are valid.

Polarion is 100% browser-based tool with support of many if not all the expected features of a modern requirement tool, including real-time collaboration, real-time communication, version history, auditability, traceability, review, approval flow and many open APIs [13].

2.3 Model based approach /mixed approach

The model based & mixed approach category consist of tools which neither can be considered document based nor notebook-based approach. These refer in general to approaches where models (e.g. architecture drawing) or a mix is used.

2.3.1 Enterprise Architect

Enterprise Architect (EA) is a tool made by Sparx Systems. It provides a wide solution for modeling, designing, analyze, and visualize entire systems. It has its focus of modeling everything, while it still supports documentation and requirements in textual formats. The tool is much more than a modeling or requirement system, it basically supports the entire life cycle of a software system [14] [15] [16]:

- Requirement management with traceability all the way through design, implementation, and testing.
- Modeled and documented through open standards like SysML, UML and others.
- Simulation of models.
- Generation of HTML documentation of entire project.
- Code generation in many programming languages from model.
- Reverse engineering of code and databases to models.
- Import/Integration of data from various tools within application lifecycle management, this includes DOORS, Jira, Confluence, and SharePoint.
- Support for many standards like AUTOSAR and IATA.
- Impact analysis of requirement changes.

2.4 Notebook approach (combined approach)

Notebook approach refers to a specific type of approach where the documentation or requirements are not written in one single document but rather spread across multiple documents which in the end are tied together by the tool used. These documents are not limited to only text and pictures but can also contain code, data and other dynamically inputs. These tools can be very simple tools with nothing more than edit and artifact (output) generation capabilities to big tools with SAT/SMT, code execution, code generation and data management.

2.4.1 Jupyter Notebook

The Jupyter Notebook is a program created by organization Project Jupyter as part of a suite of programs used for creating and sharing documents with a lot of computational data included.

Project Jupyter is non-profit and open-source organization with focus on providing free software and using open standards [17].

The notebook supports more than 40 different programming languages. It is meant for creating computational documents and not necessarily documents with a requirement-based approach. It tries to provide an interface for handling the entire process from development to result including documentation and code execution.

The Jupyter Notebook consist of two parts:

- Web application
Browser-based tool for editing the notebook document.
- Notebook documents
Document containing all content shown in the web application.

The notebook documents are saved with *.ipynb* extension to easily know the documents are Jupyter Notebook document, however the document is internally based fully on JSON.

The document is built up around cells containing code, Markdown text and raw cells. Where the raw cells are a way to avoid the nbconvert from interpret those parts of the document while it is preparing the document for presentation.

Each document, in addition to containing the code and documentation also contains all input and output from the last interactive session with the web application. This provides possibility to generate reports and show the result in a Jupyter viewer without the need of performing the interactive session again. The notebook can easily be exported to HTML, reStructuredText, LaTeX and PDF just to mention some of them [18].

Jupyter Notebook is strong in doing calculations on data and supports many languages for that purpose. It also integrates with many big data tools.

The Jupyter Notebook is quite simple and focused on data processing. It provides an interface where everyone can create extensions to improve and further expand functionality of the notebook [19].

The notebook does not in itself support any version control. One of the general solutions for handling this is to use Git. It could be plain Git tools or notebooks with Git integrated, that supports Jupyter files. On the drawback of using Git is that nearly every time committing the document it shows a lot of changes has happened while basically no changes have happened to the document. This is all due to the last interactive session stores a lot of data and that data can change from session to session [20].

2.4.2 SysML v2

SysML v2 is a system modeling language which is quite recognized within the software world which could lead to the believe that it also provided tools like Jupyter Notebook, but that is not the case. SysML v2 does not have any self-branded tool for the language, they instead suggest Eclipse and Jupyter as possible tools. Both tools support SysML v2 through extensions [21].

SysML is shortly mentioned here in the *State of the art* chapter even though it does not have its own tool and rely on other tools and therefore does not fit into the comparison. It is mentioned since SysMD aims to be compatible with SysML v2 and is not supposed to replace SysML. The fact is that SysML v2 has focus closer to modeling experts while SysMD is more focused towards the domain experts [22].

2.5 Modeling and requirement tools – Comparison

The world is filled with a lot of different tools which can be used for specifying and documenting requirements. In following table all forementioned tools are lined up and compared according to a list of criteria which has an importance for requirement and specification tools. A more detailed description of the different criteria can be seen in 2.5.2 Criteria.

2.5.1 Comparison

Comparison table of all the tools evaluated can be seen below, showing how they stack up against each other for the different criteria.

Table 1 - Comparison table between different requirement tools.

	DOORS	Excel	Polarion	Markdown	Enterprise Architect	Jupyter Notebook	SysMD Notebook
Criteria categories	Document				Model/ Mixed	Notebook	
Open-source / Open-standards	●	●	●	▲▲	▲	▲▲	▲▲
Modeling & Testing	▼	▼▼	▼	▼▼	▲▲	▼	●
Analysis functions (SAT/SMT)	●	▼	●	▼▼	▲	▼▼	▼
Traceability	▲	●	▲	●	▲	▼▼	●
Version & Change control	▲▲	●	▲▲	▼▼	▲▲	▼▼	▼▼
Tool integration	▲	●	▲▲	▼▼	▲	●	▼▼
Import	▲	▼	▲	▼▼	▲	▼▼	▼▼
Formatting, Multimedia & External files	▲	●	▲	●	▲	▼	●
Document Proofing & Generation	▲	●	▲	▼▼	▲	▼	▼▼
Collaboration	●	▲	▲	▼	▲▲	▼	▼
User interface & Usability	▲	●	▲	▼	▲	●	●

▲▲ Very good ▲ Good ● Average ▼ Below average ▼▼ Bad

The comparison is based on the authors personal experience in use of most of the tools (Excel, Polarion, Markdown, SysMD Notebook) while rest of tools evaluation is based on research. This difference in evaluation and no peer review can cause an unwilling bias towards some

tools. This needs to be taken into consideration in when referencing this comparison. The basis for the evaluation can be found in Appendix: B Tools comparison evaluation.

Comparison results & conclusion The results from the comparison table shows the big commercial tools coming out on top in general. The big commercial tools in general performs well on all parameters and clearly outperform other tools in traceability, version control, and change management. All the other tools in this comparison does not seem to care about traceability, version control and change management and lets the user manage that through third party tools.

This comparison has focused on the technical aspect of the tools and have not considered neither the running cost, upfront cost, nor the users time.

When looking at SysMD, it does in general not perform that well compared to the other tools. In analysis it neither perform that well except for SAT/SMT area where it probably performs the best of all the tools. Polarion and DOORS performs ok in analysis area, however this is more in textual parts rather than analysis of modeling parts. EA does very well in modeling and testing which is due to its deep focus on simulation of the models.

From a modeling perspective EA comes out on top with SysMD following. The two tools' targets two different user groups, with EA having a focus on modeling experts and SysMD focusing on the domain experts which does not necessarily have deep modeling experience.

From a requirement perspective Polarion and DOORS comes out on top with EA following. SysMD lacks behind on many parameters within this area but wins on the analysis part of specification and modeling. E.g. Consistency checking.

From a mixed documentation, modeling, and requirement perspective it is probably EA that wins followed by Polarion, DOORS, and then SysMD. They all have their pros/cons within this category, where SysMD is probably the simplest of them all.

2.5.2 Criteria

Criteria for the comparison table are selected based on different sources specifying what are important for a requirement tool [23] [24].

Open-source / open-standards The open-source & open-standards category evaluates the tools based on how friendly they are towards open-source and open-standards.

It looks if the tool is open-source if the data is stored in open-standard document format or available through open-standard APIs.

Modeling & testing Modeling & testing category relates to how well the tool supports modeling of requirements to:

- Visual representation.
- Standard modeling language like SysML or UML.

Testing part looks at the possibilities for verification and validation by use of various tools for testing. Here the focus is on individual requirements and not the complete project.

Analysis functions (SAT/SMT) The analysis functions category focuses a lot on the satisfiability modulo theories (SMT) and the Boolean satisfiability problem (SAT) for how well tools are to validate and verify whether there are conflicting statements and requirements in the documentation.

Another part of the analysis is the functionality to:

- Scan text for unsuitable, inexact, and wrongly used language or terminology.
- Analyze project for inconsistencies, missing links, or gaps in traceability.

The first part consists of the ability of to detect issues and the other part is about how well it reports found issues to the user.

Traceability This category is about how the tools support traceability between different requirements and different stages of a project.

- User friendliness for linking of requirements.

- Linking possibilities consisting of attributes, objects, and linking directions.
- Navigation, overview, and tracing of links.

Version & change control The version & change control category focuses on both direct and in-direct version and change control possibilities which the tool supports. This consist of two overall parts: version control and change control.

In version control:

- Possibilities to track any changes happened to project including information about who did a change and when it happened.
- Track changes between different released versions.
- User friendliness of using the version control.

For tools not including their own version control then in-direct version control using GIT/SVN is considered in evaluation of the tool.

In change control:

- Review of changes before implemented in project.
- Controlled process for updating documents to avoid accidentally unwanted changes.
- Comment and discussion tracking for each change.

Tool integration This category evaluates the tool in relation to integration of other tools functionality into the requirement tool and the possibility of including external tools into the requirement tool.

Tool integration can both be considered integration of other tools or linking object/requirements to information in other tools.

Import This relates to the capabilities of importing documents and requirements from existing documents (e.g. MS Word, PDF, Excel) or from other tools.

The evaluation is based on how well it recognizes text, structures, formatting, etc. and converts this correctly into the tool's native language.

Formatting, multimedia & external files This evaluation category evaluates tools capability for including documents and files of other file types than the native file type of the tool. E.g. If the tool uses Markdown file format, then MS Word, PDF and JPG would as examples be considered non-native file formats.

The formatting part rates the tool in relation to how much of document formatting is possible to do and how easy handling the formatting is.

Document proofing & generation Document proofing and generation category evaluates tools capabilities from the point of view of generating documents and views for internal and external use. This includes:

- Visual views of the project.
- Generation of documents for use both internal and external (customers, suppliers). These documents could both contain everything but also a sub-set of the project.
- Proofing capabilities to check for spelling, grammar, and style guide.

Collaboration The collaboration category evaluates the possibilities of having more than one user working together at the same project.

- Multiple users working at the same time on the project.
- Lock sections of project to prevent other users from changing.
- Live tracking of what other users is working on.

User interface, views & usability This category focus on the user experience of the tool. This includes everything from how easy the tool is to use to how many ways the project information can be shown.

- Views is about how many ways the information can be represented. Regular document view, model views based on inheritance or relationships, and graphical views.
- Usability of tool, how easy it is to use, is it necessary to read the manual just to use the tool.
- Whether functionality in user interface is intuitive to use or not.

2.6 SysMD tool – Deep dive

SysMD is both a tool and a language inspired by SysML v2. The tool is aimed at domain experts with minimal high level modeling expertise. SysMD as a language is inspired by SysML v2, however designed in a way to use intuitive and near natural-language statements to make it easy for users to use [25]. Ensuring minimal learning curve for any to read documents and simple syntax for contributors to learn.

SysMD consist of three parts: Language, Recommender System, and SysMD Notebook. The notebook comes in both an app and web service version which in short are the same. Main difference is that the app works in both offline and online mode, while web service is only online mode.

2.6.1 SysMD language

SysMD language syntax is designed with a primary focus in mind - the target audience. The target audience is domain experts as companies would like to avoid having to hire modeling experts for the pure purpose of doing specification work. Based on these requirements the main requirements for the language are:

- SysMD needs to be easy to use without modeling experience.
- SysMD needs to be easy to read without modeling experience.
- SysMD must help contributor to keep models and systems connected and well documented.
- SysMD should be compatible with SysML v2.

SysMD language syntax is designed to use short, easy statements written in natural English language. Use of the natural language helps domain experts writing easy to understand statements without the need of learning a complicated language.

SysMD syntax and meta model is closely inspired by SysML v2 as there is a wish for being interoperable with SysML v2, but to no degree replacing SysML v2 as SysMD is lacking many features from SysML v2.

SysMD is on purpose kept simple and each statement in SysMD corresponds to a single entry in the meta model. A statement consists of *Subject Predicate Object* “. ”. Where *Object* can be even further defined with arithmetic constraints or dependencies. Many statements mainly describe the relationships it has to other statement in the SysMD model.

The pre-defined predicates are: *isA*, *hasA*, *satisfies*, *implements*, and *executes*.

The most important relationships used in the SysMD language is the pre-defined *isA* and *hasA*. Both is used together to describe the model, where both shapes a tree which is laid across each other in which they create the main part of the project model.

The *isA* models the inheritance and describes what something is. The further break-down into parts is done with de-composition, which consist of following keywords *hasA*, *defines*, and *imports*. Each of these keywords has its own meaning in relation to de-composition of the object [25].

hasA *hasA* describe the occurrence or instantiation of some elements in an object. These elements can be of type property or instantiation. The syntax is very similar for both property and instantiation, while property always describes an item of class *Value* then instantiation describes items of other class types.

defines *isA* is the predicate in a semantic triple and always declares a new class where the subject is an identification which has not been used yet, and the object is identification that refers to another definition.

defines adds a list of semantic triples as definitions to an element given by a subject. These definitions are of either *isA* relation to classes or user defined relations.

imports *imports* are a feature in SysMD language to add other elements into the scope of a specific element. In this way elements from other parts of the project can become “visible” and thereby be used as relationship within the element. *imports* itself does not add any direct relationship between given elements, it only relates to the visibility of the element to enable possibility of creating relationships with *isA* and *hasA*.

Imports works on both *Namespace* and *Document* in either case they are included into the current document. In case of *Document* then all elements of the Global *Namespace* are included into the current document [26].

In SysMD Notebook a document refers to a separate Markdown file within a project.

Example of how the above forementioned syntaxes all plays together can be seen in code example below. Here the class *Types* has been declared in another md file than *Product* class, however *Product* class is still able to refer to *Types* because of the use of *imports*.

```
//----- Start of Types.md file -----//
Types isA Component.
Types defines
  HwType isA Function.

Types::HwType defines
  TypeA isA Component.
//----- End of Types.md file -----//

Document imports Types.

Product isA Package.
Product defines
  ProdA isA Component.

Product::ProdA hasA Component id: Types::HwType::TypeA.
```

2.6.2 SysMD Recommender System

As part of the SysMD tool environment there is a system called Recommender System which purpose is to help the user reuse existing known elements rather than creating new elements.

The Recommender System is a separate part of the SysMD environment consisting of Recommender System and Agila backend, both is not needed for using SysMD but is a helping tool to improve usage of SysMD. Agila backend is an online database which have the purpose of storing all elements created by the users of SysMD, which means every time a user creates an element in SysMD that element is then stored in the database. The main idea behind this is to enable users to reuse already existing elements, and thus saving time not having to create same elements again [25].

The reuse idea will only work if it is easy for the user to find existing elements else user will end up creating new element and this is where Recommender System comes into play. Recommender System works in a way where it receives the new element that the user wants to create, it searches the Agila backend for similar existing elements using a “Semantic Search Engine” to return all elements that has a meaningful relevance to the element which the user wishes to create. In this way the Recommender System will nudge the user to use an existing element before creating a new one. See Figure 6 below for Recommender System usage.

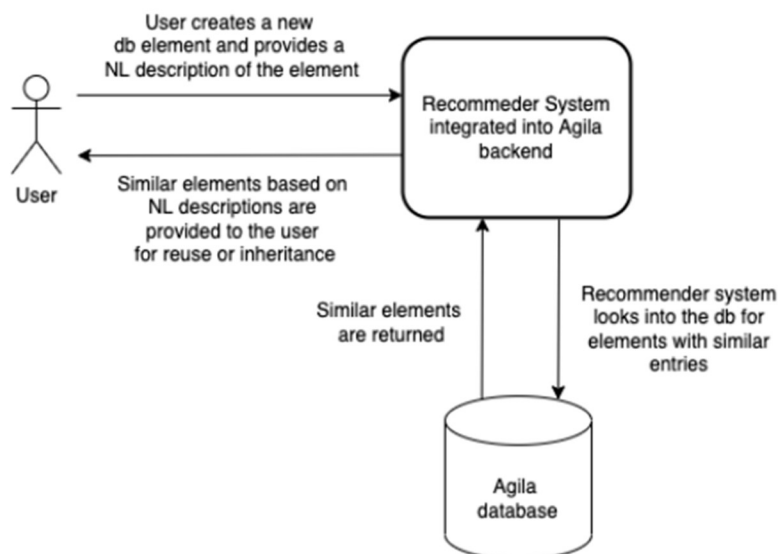


Figure 6 - Agila Recommender System in action [25].

2.6.3 SysMD Notebook

SysMD Notebook is part of the SysMD tool environment and is one of the tools that ties everything together. This tool comes in both an app and web service version and are currently the only tools that officially supports SysMD language and have analysis functions built up around it to improve the value of SysMD.

Markdown files The SysMD Notebook uses only Markdown files for documentation of projects. These md files are constructed in a special way to both contain documentation and models in same md file [22].

- Documentation is written in a standard Markdown format.
 - Supporting all the common things: formatting, pictures, tables, figures, links.
- Models are written in SysMD language (using fenced code blocks).

Even though SysMD Notebook uses the md files in a slightly special way, the md files are still fully compatible with regular Markdown viewers. SysMD Notebook places a tag *SysMD* in front of each code block to specify which blocks in the md file should be treated as model elements (SysMD language) in the Notebook. See *VS Code: Text view* picture in Figure 7 for the code block tag.

Example of Notebook document shown in Visual Studio Code (VS Code) and SysMD Notebook can be seen on Figure 7. The document is perfectly shown in VS Code however missing the syntax coloring which is present in SysMD Notebook.

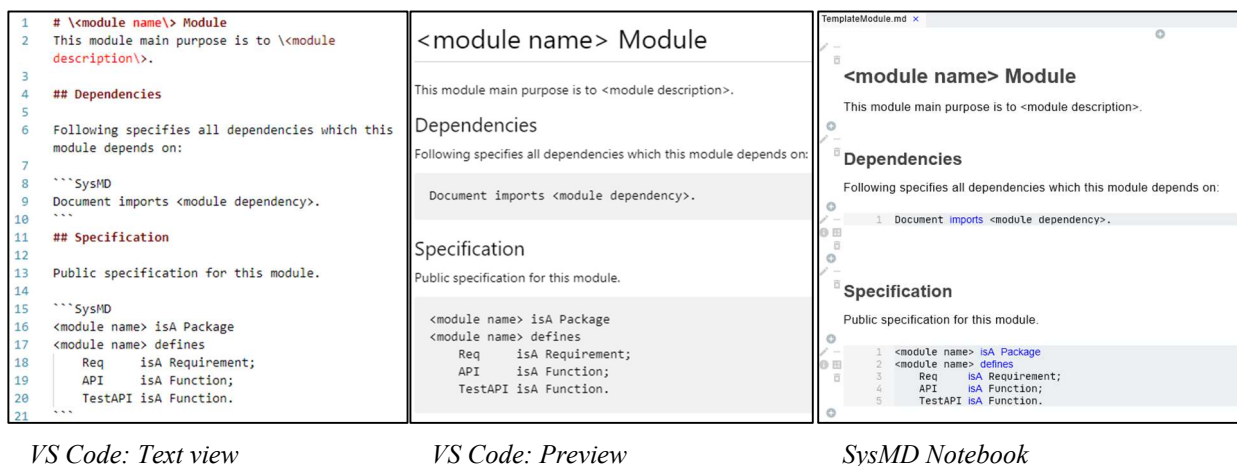


Figure 7 - Example of SysMD document in Visual Studio Code and SysMD Notebook.

Analyze functionality SysMD Notebook support several different analysis functionalities to ensure a user friendly experience of using SysMD.

- Syntax check
- Constraint check
- *hasA* and *isA* relationship tree-views

Syntax check is there to ensure the user has written the models correctly according to the SysMD language syntax thus ensuring the models can be analyzed and computed correctly.

Constraint check helps ensuring that there are no requirement constraints that are inconsistent throughout the models. E.g. superclass *Test* specifies that $A = [2 .. 10]$ and class *TestA* specifies that $A = [1 .. 5]$. In this case it will give inconsistency in *TestA* because it says value $A = 1$ is valid while *Test* has specified it cannot be less than 2. See example in Figure 8 below.

The screenshot shows a SysMD Notebook interface with a code editor on the left and an 'Agenda' panel on the right. The code editor contains the following code:

```

1 SuperTest isA Package.
2 SuperTest defines
3   Test isA Component;
4   TestA isA Test.

1 SuperTest::Test hasA
2   Value A: Real(2 .. 10).
3
4 SuperTest::TestA hasA
5   Value A: Real[1 .. 5].

SuperTest::Test::A = 2..10
SuperTest::TestA::A = *..*
ERROR in SuperTest::TestA::A: INCONSISTENCY: subclass value Real
of A must be refinement of superclass value 2..10

```

The 'Agenda' panel on the right shows a summary of 1 item in the agenda and a red warning icon with the following message:

INCONSISTENCY:
subclass value Real of A
must be refinement of
superclass value 2..10

Figure 8 - Constraint check example.

The tree-views consist of an *isA* and *hasA* tree-view, they are there for ensuring the user visually can see whether the relationships created from the models looks as expected.

isA shows the element type relationship providing possibility to trace which classes inherits from a specific class type. The *hasA* visualizes the models and which classes a specific model consists of and how they relate. See examples of *isA* and *hasA* tree-views respectively in Figure 9 below.

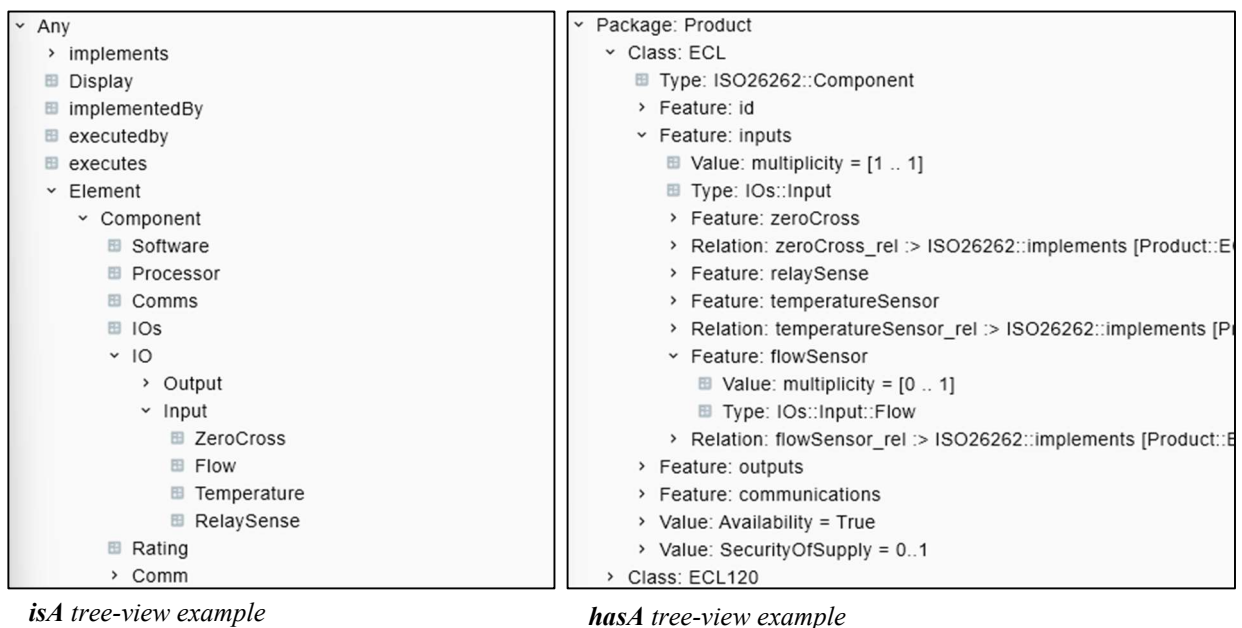


Figure 9 - Example of *isA* and *hasA* tree-views in SysMD Notebook.

3 SysMD in industrial application

The following chapters is about evaluation of the case study SysMD for use in industrial application. The evaluation focuses on how intuitive SysMD, as a language and tool, is to use and how well it handles the challenges of a real world example. In this case study the ECL product family from Danfoss, have been used for the evaluation of the SysMD tool.

3.1 Project description for case study

This project in SysMD is based on the ECL product family. It consists currently of two individual products ECL 120 and ECL 220 which together creates what is called the ECL x20 platform. As of writing the ECL 120 has just been released (2022-Q3) and the ECL 220 is scheduled for release later in 2023.

The ECL x20 platform is constructed based on multiple platforms and part of it can be seen in Figure 10. This SysMD project focus on the ECL x20 software platform excluding the MBD application software (see highlighted parts of platform picture). It does not consider other parts of the ECL x20 platform.

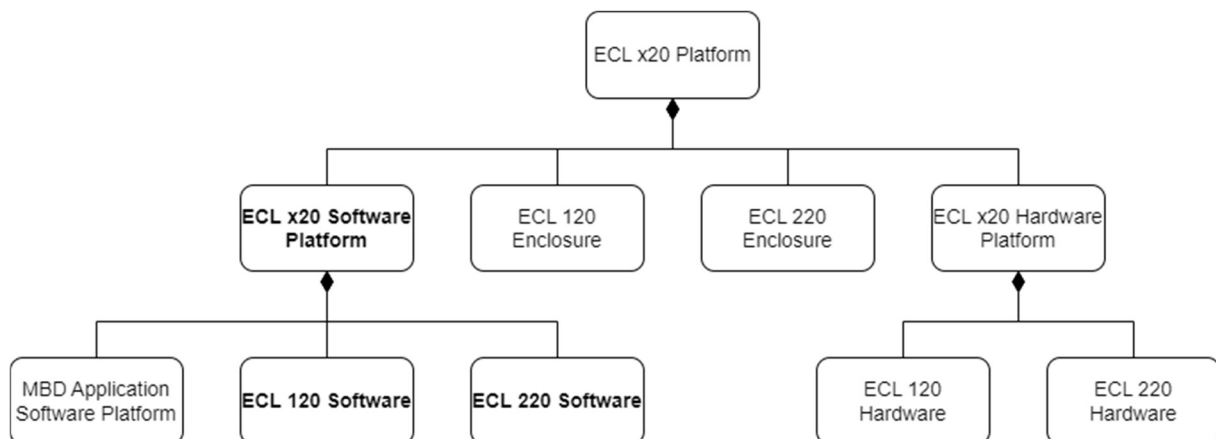


Figure 10 - Overview of ECL x20 Platform.

The focus on the software platform has been chosen due to that having the most direct potential to use SysMD for various reasons.

- Software is in general the most evolving part during the product lifecycle, and documentation often gets outdated and become inconsistent with new requirements.
- Modeling of platform and requirements could directly result in auto-generated code.
- Modeling of requirements could improve development of test cases for automatic tests.
- Enclosure and hardware are generally much more stable after release of the product which results in much more stable and correct documentation that already fulfills requirements.

3.1.1 Focus

This case study is limited in calendar time which gives a conflicting focus between what would benefit the ECL project the most and what is most beneficial for the SysMD case study.

The ECL project would like to have complete documentation in all cases, this also applies to all subparts which a project like the ECL can be split into. E.g. if hardware inputs of ECL are documented then it should be all inputs and not just a subset. In this case the case study only needs enough inputs documented to validate the concept works and covers various scenarios of modeling and documentation.

In all cases of conflict the case study needs have taken precedent to ensure modeling and documentation in SysMD covers as many different scenarios as possible.

3.1.2 ECL product family

The ECL product family of concern for this case study consist of two products, ECL 120 and ECL 220, which is developed based on the same hardware and software platform, ECL x20 Platform. Visualization of the two products can be seen on Figure 11.



Figure 11 - Visualization of ECL 220 and ECL 120.

Both products can be used for controlling various applications, main focus is on applications controlling heating of a house or apartment based on district heating. Example of an application circuit can be seen in Figure 12.

In top center of the drawing the ECL controller can be seen connected with wires to various temperature sensors, pumps, and valves for controlling the heating to both radiator and domestic hot water in a house or apartment.

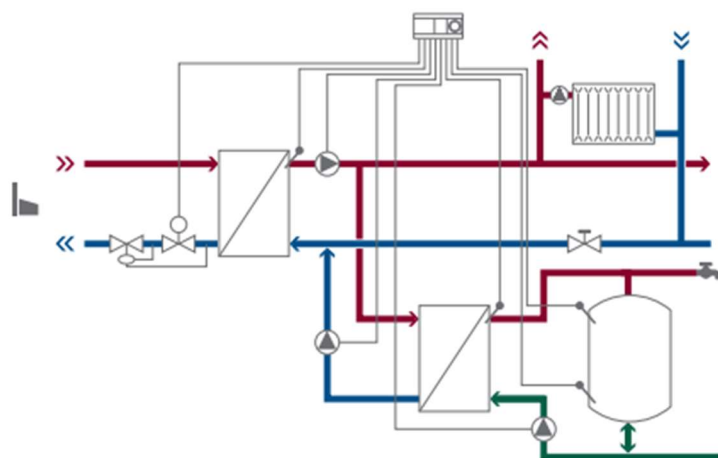


Figure 12 - Example of flat station application with radiator and domestic hot water.

The main difference between the different ECLs is the number of IOs they have, see Table 2. The varying number of IOs enables the larger ECLs to support a broader variety of different applications where more IOs is required.

Table 2 - IOs comparison table of ECLs.

IOs	ECL 120	ECL 220
Inputs		
<i>Temperature Sensor PT1000</i>	4	5
<i>Flow Sensor</i>	0	1
<i>Potential Free Sense Input</i>	1	1
Outputs		
<i>Relay</i>	1	2
<i>PWM Pump</i>	1	1
<i>Stepper Motor</i>	0	2
<i>2-Point / 3-Point valves Triac</i>	2	2
Communication		
<i>Ethernet</i>	1	1
<i>Modbus RTU RS485</i>	1	1
<i>BLE</i>	1	1
<i>Zigbee</i>	1	1
<i>M-Bus</i>	0	1

3.2 Project in SysMD

The ECL x20 Software Platform project in SysMD is completely generated through the SysMD Notebook. The project is constructed in SysMD Notebook with a hierarchical structure in mind which also adhere to the concept of “low coupling and high cohesion”.

The project in SysMD Notebook consist of several documents of the filetype md. With each document containing project documentation in Markdown language and models in SysMD language. All md files can be found as complete files in *Appendix: A SysMD project files*.

The project has been developed with following software versions of SysMD Notebook: 2.7.9 to 2.9.10

3.2.1 Project structure

The project structure in SysMD Notebook is as following:

- ECL.md
 - Product.md
 - RS485.md
 - FlowSensor.md
 - TemperatureSensor.md
 - ZCD.md
 - Types.md
- Templates/
 - TemplateModule.md

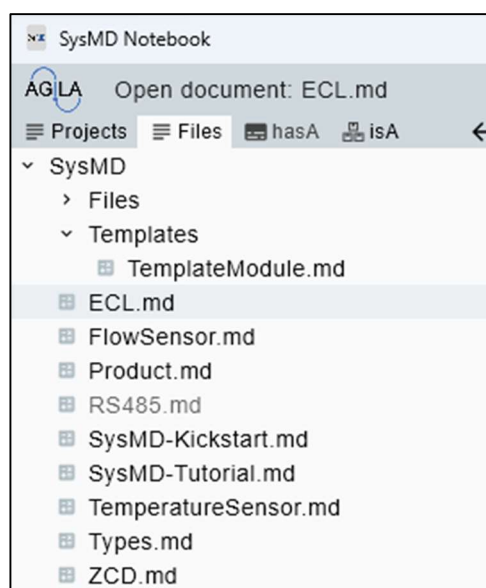


Figure 13 - Project structure in SysMD Notebook.

ECL.md is meant to be the main document that links everything else together. It does not contain any models and is purely meant to be the introduction for the ECL x20 Software Platform. The project structure in SysMD can be seen in Figure 13.

Product.md is describing the products from their IOs point of view (Inputs, Outputs, Communication) and each type of IO (module) is described/modelled in their own md file. These files contain detailed information about their requirements and public interfaces.

Types.md is describing product and platform generic types which are used across the entire project and therefore not bound to a specific module.

Templates folder contains md files which is created for the sole purpose of functioning as templates for when new md files must be created. Currently only *TemplateModule.md* exist which is used for when a new module needs to be created.

3.2.2 Product overview

The product overview which shows the IO differences between the ECL 120 and ECL 220 is constructed in a few sections:

1. Definition
2. ECL product family description
3. Variant description

Definition This part consists of dependencies, type definitions and available IOs.

Dependencies mainly consist of importing all the modules for the different IOs, which is done with *Document imports* keyword to import a given document into this document [26].

```

1 Document imports ScalarValues.
2 Document imports Gbo.
3
4 Document imports Types.
5 Document imports ZCD.
6 Document imports TemperatureSensor.
7 Document imports FlowSensor.
8 Document imports RS485.
```

Figure 14 - Document imports in Product Overview.

Type definitions specifies the ECL product family and all its variants; ECL, ECL 120, and ECL 220. This is the main part of the document where *Product* is specified as the top level.

```

1 Product isA Package.
2 Product defines
3   ECL isA Component;
4   ECL120 isA ECL;
5   ECL220 isA ECL.
6
7 Product::ECL hasA Component id: Types::HwType::ECL.
8 Product::ECL120 hasA Component id: Types::HwType::ECL120.
9 Product::ECL220 hasA Component id: Types::HwType::ECL220.
```

Figure 15 - Product and variant definition.

Last part is the available IOs which specifies all the possible inputs, outputs, and communication interfaces that an ECL can have. All this together is the foundation for being able to specify the ECL product family and its variants.

```

1 IOs isA Component.
2 IOs defines
3   IO      isA Component;
4   Input   isA IO;
5   Output  isA IO.

- List of available inputs.

1 IOs::Input defines
2   Temperature isA Component;
3   RelaySense  isA Component;
4   ZeroCross   isA Component;
5   Flow        isA Component.

- List of available outputs.

1 IOs::Output defines
2   Relay      isA Output;
3   Triac     isA Output;
4   Stepper   isA Output;
5   PumpPWM   isA Output.

- List of available communication interfaces.

1 Comms isA Component.
2 Comms defines
3   Comm      isA Comm;
4   Ethernet  isA Comm;
5   RS485     isA Comm;
6   MBus      isA Comm;
7   Zigbee    isA Comm;
8   Serial    isA Comm;
9   BLE       isA Comm.

```

Figure 16 - Specifying list of available types of IOs.

ECL product family There are different ways to describe product families and the specific product variants. For the ECL product family it is chosen to specify the base as a full list of all possible configuration including all product variants which can be seen in Figure 17.

The specification specifies which inputs, outputs, and communication types are available in the family. It at the same time also specifies the minimum and maximum number of possible instances of a certain IO.

```

1 Product::ECL hasA
2   Component inputs:           IOs::Input;
3   Component outputs:          IOs::Output;
4   Component communications:    Comms.
5
6 Product::ECL::inputs hasA
7   Component zeroCross:        [1 .. 1] IOs::Input::ZeroCross;
8   Relation zeroCross_rel:     zeroCross implements ZCD;
9   Component relaySense:       [0 .. 1] IOs::Input::RelaySense;
10  Component temperatureSensor: [0 .. 5] IOs::Input::Temperature;
11  Relation temperatureSensor_rel: temperatureSensor implements TemperatureSensor;
12  Component flowSensor:        [0 .. 1] IOs::Input::Flow;
13  Relation flowSensor_rel:     flowSensor implements FlowSensor.
14
15 Product::ECL::outputs hasA
16  Component relay:              [0 .. 2] IOs::Output::Relay;
17  Component pumpPwm:           [0 .. 1] IOs::Output::PumpPWM;
18  Component triac:             [0 .. 2] IOs::Output::Triac;
19  Component stepper:           [0 .. 2] IOs::Output::Stepper.
20
21 Product::ECL::communications hasA
22  Component ethernet:          [1 .. 1] Comms::Ethernet;
23  Component rs485:             [0 .. 1] Comms::RS485;
24  Relation rs485_rel:          rs485 implements RS485;
25  Component serial:            [1 .. 1] Comms::Serial;
26  Component ble:               [1 .. 1] Comms::BLE;
27  Component zigbee:            [0 .. 1] Comms::Zigbee;
28  Component mbus:              [0 .. 1] Comms::Mbus.

```

Figure 17 - ECL product family specification.

Besides the specification specifying the IOs and number of instances of each IO, it also specifies the type of each IO. This is done using relationship using the *Relation* and *implements* syntax [26]. The code example below shows how to create a relationship using as an example the TemperatureSensor module.

```

Document imports TemperatureSensor.
...
IOs::Input defines
  Temperature isA Component;
...
...
Product::ECL::inputs hasA
  Component temperatureSensor: [0 .. 5] IOs::Input::Temperature;
  Relation temperatureSensor_rel: temperatureSensor implements TemperatureSensor;
...
...

```

ECL Variants The ECL variants, ECL 120 and ECL 220, are both considered classes of the superclass ECL, therefore both variants will initially inherit the exact same specification as the ECL base class. For fixing this and ensuring the variant only has what it is supposed to have, there are two different approaches to achieve this:

1. Only specifying what is different from base.
2. Specifying everything as in base but overwriting the number of instances of each IO to the correct number.

Any approach has its own pros/cons, and in this project approach number 2 was selected due to the value of having everything specifying a variant collected in one place. The implementation for ECL 120 looks as following (Figure 18).

```

1 Product::ECL120::inputs hasA
2   Component zeroCross:      [1 .. 1] IOs::Input::ZeroCross;
3   Component relaySense:     [1 .. 1] IOs::Input::RelaySense;
4   Component temperatureSensor: [4 .. 4] IOs::Input::Temperature;
5   Component flowSensor:     [0 .. 0] IOs::Input::Flow.
6
7 Product::ECL120::outputs hasA
8   Component relay:          [1 .. 1] IOs::Output::Relay;
9   Component pumpPwm:       [1 .. 1] IOs::Output::PumpPWM;
10  Component triac:          [2 .. 2] IOs::Output::Triac;
11  Component stepper:        [0 .. 0] IOs::Output::Stepper.
12
13 Product::ECL120::communications hasA
14  Component ethernet:       [1 .. 1] Comms::Ethernet;
15  Component rs485:          [1 .. 1] Comms::RS485;
16  Component serial:         [1 .. 1] Comms::Serial;
17  Component ble:            [1 .. 1] Comms::BLE;
18  Component zigbee:         [1 .. 1] Comms::Zigbee;
19  Component mbus:           [0 .. 0] Comms::Mbus.

```

Figure 18 - ECL 120 IO specification in SysMD.

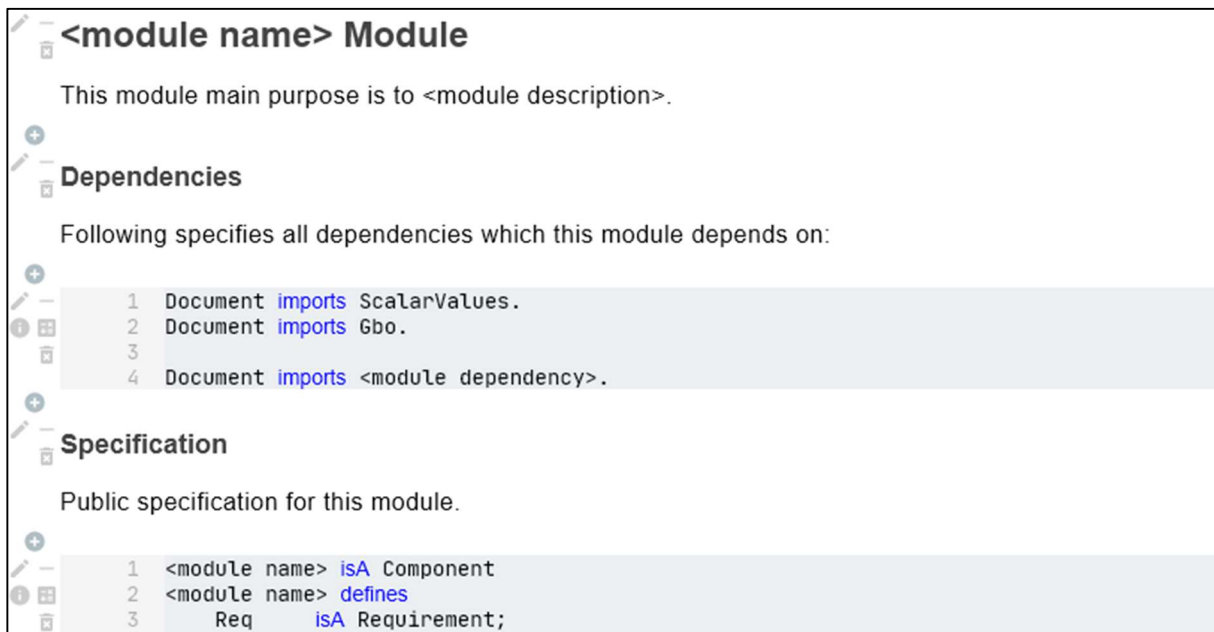
The ECL 120 specification looks much like the base specification with small changes. The number instances of each IO are now specific, and relationships to the IO modules are not shown here as they are already inherited from the ECL superclass. It furthermore shows which modules are not part of the product but are still supported by the family.

3.2.3 SysMD templates

To ease the creation of new modules and ensuring they all have a similar structure a template md file has been created which helps to construct new modules in similar way. The steps for creating the initial part of a new module:

1. Create copy of *TemplateModule.md*
2. Rename file to name of new module.
3. Replace *<module name>* in entire file with the name of the new module.
4. Replace *<module description>* with description of what this new module is supposed to do.
5. Now the new module has been created and proper information needs to be filled into the sections *Requirements*, *Public API*, and *Test API*.

The template module file starts like it can be seen on Figure 19.



```

<module name> Module
  This module main purpose is to <module description>.

  Dependencies
  Following specifies all dependencies which this module depends on:
  1 Document imports ScalarValues.
  2 Document imports Gbo.
  3
  4 Document imports <module dependency>.

  Specification
  Public specification for this module.
  1 <module name> isA Component
  2 <module name> defines
  3   Req isA Requirement;
  
```

Figure 19 - Start of *TemplateModule.md* file.

3.2.4 Modules

For showing some of the steps in module implementation the modules *TemperatureSensor* and *RS485* are used as examples.

Each module follows the module template, see Figure 20. Firstly it can be noticed that *<module name>* has been replaced with *TemperatureSensor* and there has been added an overall natural language description to the document.

Types has been imported as the temperature sensor module needs some generic project types.

Temperature Sensor Module

ECL 120 and ECL 220 has each temperature sensor inputs. These are used to measure relevant temperatures for the application whether this being temperature sensors internally in the flat station or being room/outdoor temperature sensors.

These temperature sensors must be measured at regular interval and with good repeatability to ensure the usability of the MBD regulation applications.

Dependencies

Following specifies all dependencies which this module depends on:

- 1 Document `imports` `ScalarValues`.
- 2 Document `imports` `Gbo`.
- 3
- 4 Document `imports` `Types`.

Specification

Public specification for this module.

- 1 `TemperatureSensor` `isA` `Component`.
- 2 `TemperatureSensor` `defines`
- 3 `Req` `isA` `Requirement`;
- 4 `API` `isA` `Function`;
- 5 `TestAPI` `isA` `Function`.

Figure 20 - Start of temperature sensor module.

The next section of each module document is the requirement specification, where all the requirements this module needs to fulfill, are specified. Firstly, are all top level groups of requirements defined, before they are further detailed. In case a requirement is not clearly detailed, it is possible to add comments. Examples of this can in the following be seen from temperature sensor (Figure 21) and RS485 (Figure 22) modules.


```

Requirements
Following section describes the specific requirements there is for the temperature sensor component.

1 TemperatureSensor::Req defines
2   Range isA Requirement;
3   AccurateRange isA Range;
4   SampleRate isA Requirement;
5   SupportedSensor isA Requirement;
6   NoiseImmunity isA Requirement.
7
8 TemperatureSensor::Req::Range hasA
9   Value range: Real(-64 .. 192) [°C];
10  Value accuracy: Real(-5 .. 5) [K];
11  Value resolution: Real(0 .. 0.1) [K];
12  Value repeatability: Real(-1 .. 1) [K].
13
14 TemperatureSensor::Req::AccurateRange hasA
15  Value range: Real(-10 .. 95) [°C];
16  Value accuracy: Real(-2 .. 2) [K];
17  Value resolution: Real(0 .. 0.01) [K];
18  Value repeatability: Real(-0.1 .. 0.1) [K].

```

Figure 21 – Example of some requirements from temperature sensor module.

Requirements can be a further refinement of another requirement which can be seen from *Range* and *AccurateRange*.

```

1 RS485::Req defines
2   Interface isA Requirement;
3   Protocol isA Requirement;
4   Modbus isA Requirement.
5
6 RS485::Req::Interface hasA
7   Value baudrate: Integer(9600 .. 115200); // 9600, 19200, 38400, 57600, 76800, 115200
8   Value parity: Integer(0 .. 2); // 0 = None, 1 = Even, 2 = Odd
9   Value bias: Boolean. // False = Off, True = On
10
11 RS485::Req::Protocol hasA
12  Value supportModbus: Boolean(true); // True = Supporting Modbus
13  Value supportBACnet: Boolean. // True = Supporting BACnet
14

```

Figure 22- Example of some requirements from RS485 module.

The purpose of the last section of the module is to document the public API. The public API has from a modeling point of view a great future potential. In future it can be used to auto-generate code for the modules public API.

SysMD does not have any syntax in its language to support direction of data, whether a function argument is an input, output, or both. To handle this in the models these prefixes: *in*, *out*, *inout*, and *ret*, is used on the elements of a function.

- *in* specifies that argument only provides data to function.

- *out* specifies that argument returns data from function, this is usually a pointer in a language like C.
- *inout* specifies that argument both provides data to function, but function also returns data.
- *ret* specifies the return value from the function.

Example of this can be seen in Figure 23 below.

```

1 TemperatureSensor::API defines
2   TempSensor_IsSupported           isA Function;
3   TempSensor_GetTemperature       isA Function;
4   TempSensor_SetRTD               isA Function;
5   TempSensor_ApplyCalibrationValue isA Function;
6   TempSensor_Handler              isA Function;
7   TempSensor_Init                 isA Function.
8
9 TemperatureSensor::API::TempSensor_IsSupported hasA
10  Value inSensorId: Integer = TemperatureSensor::Req::SupportedSensor::maxNumOfSensor;
11  Value retIsSupported: Boolean.
12
13 TemperatureSensor::API::TempSensor_GetTemperature hasA
14  Value inSensorId: Integer = TemperatureSensor::Req::SupportedSensor::maxNumOfSensor;
15  Value outStatus: Integer(0 .. 3);
16  Value outValue: Real = TemperatureSensor::Req::Range::range;
17  Value retValidData: Boolean.

```

Figure 23 - Example of function specification in temperature sensor module.

3.2.5 Analysis with SAT/SMT solver in relation to consistency.

SysMD support SAT/SMT analysis specifically within the area of consistency. Example of this in the ECL project can be seen in the measurement range specification in the temperature sensor module.

The full measurement range, *Range*, is specified to be from -64°C to 192°C while the important accurate range, *AccurateRange*, is specifying the range to be from -10°C to 95°C.

```

TemperatureSensor::Req defines
  Range           isA Requirement;
  AccurateRange  isA Range.
...
TemperatureSensor::Req::Range hasA
  Value range:    Real(-64 .. 192) [°C].
...
TemperatureSensor::Req::AccurateRange hasA
  Value range:    Real(-10 .. 95) [°C].

```

As *AccurateRange* is a subclass of *Range* then its refinement of measurement range must be within *Range* specification. In case for any reason the *AccurateRange* does not stay within *Range* specification SysMD will report an inconsistency error.

E.g. *AccurateRange* was accidentally specified to -100°C instead of -10°C , this error will be catch by SysMD when the analyze is performed.

```

1 TemperatureSensor::Req defines
2   Range isA Requirement;
3   AccurateRange isA Range;
4   SampleRate isA Requirement;
5   SupportedSensor isA Requirement;
6   NoiseImmunity isA Requirement.
7
8 TemperatureSensor::Req::Range hasA
9   Value range: Real(-64 .. 192) [°C].
10
11 TemperatureSensor::Req::AccurateRange hasA
12   Value range: Real(-100 .. 95) [°C].
13
14
TemperatureSensor::Req::Range::range = -64..192 °C
TemperatureSensor::Req::AccurateRange::range = -100..95 °C
ERROR in TemperatureSensor::Req::AccurateRange::range: INCONSISTENCY: subclass value
-100..95 of range must be refinement of superclass value -64..192

```

Figure 24 - Example of inconsistency check.

3.2.6 Computation of requirements

SysMD supports the possibility of performing computations on elements to do computational refinements for further refining requirements. This is done among others with a set of predefined functions in the SysMD language. One of these functions is *ITE*, an “if ... else ...” function, a use case for this can be seen in the RS485 module.

The Modbus specification for RTU message framing on RS485 states following [27]:

- T3.5: At least 3.5 character times between frames on bus.
- T1.5: Max 1.5 character times between two characters in frame for valid frames.
- For baud rates above 19200Bps fixed values should be used for the two timeouts.
 - T1.5 = 750μs
 - T3.5 = 1.750ms

The Modbus specification is very clear about the T1.5 and T3.5 timeouts, however the actual value range is highly dependent on the supported baud rates in the specific product. Therefore from a modeling point of view the T1.5 and T3.5 needs to be calculated before they can be useful. This is here the *ITE* function comes into play. The pseudo syntax for calculating the T1.5 and T3.5 will be something like this where X represents either T1.5 or T3.5:

```
RS485::Req::Modbus hasA
Value tXTimeout: Time(0.5 .. 5.0) [ms]
= ITE(Baudrate > 19200, XTimeoutFixed, X / Real(Baudrate / BitSize)).
```

The time range specified for *tXTimeout* will automatically be calculated based on the formula in *ITE*.

Output for the real implementation of the T1.5 and T3.5 including calculations can be seen in Figure 25 below.

```

1 RS485::Req defines
2   Interface isA Requirement;
3   Protocol isA Requirement;
4   Modbus isA Requirement.
5
6 RS485::Req::Interface hasA
7   Value baudrate: Integer(9600 .. 115200). // 9600, 19200, 38400, 57600, 76800, 115200
8
9 RS485::Req::Modbus imports SI.
10 RS485::Req::Modbus hasA
11   Value charBitSize: Integer(11 .. 11);
12   Value t1_5Timeout: Time(500.0e-3 .. 2.0) [ms]
13     = ITE(RS485::Req::Interface::baudrate > 19200, 750e-3[ms], 1.5 / Real(RS485::Req::Interface::baudrate / charBitSize));
14     // Inter character timeout on bus. Above 19200 = 750us
15   Value t3_5Timeout: Time(1.50 .. 5.0) [ms]
16     = ITE(RS485::Req::Interface::baudrate > 19200, 1.750[ms], 3.5 / Real(RS485::Req::Interface::baudrate / charBitSize)).
17     // Inter frame timeout on bus. Above 19200 = 1.750ms

```

```

RS485::Req::Interface::baudrate = [9600 .. 115200]
RS485::Req::Modbus::charBitSize = [11 .. 11]
RS485::Req::Modbus::t1_5Timeout = 0.5..2 ms
RS485::Req::Modbus::t3_5Timeout = 1.5..3.86058 ms

```

Figure 25 - T1.5 and T3.5 timeouts implementation in SysMD with calculated results.

Those with sharp eyes might notice that SysMD are not calculating the time ranges correctly for T1.5 and T3.5. The correct values:

- T1.5 = 0.750ms to 1.719ms (not *0.5ms to 2.0ms*).
- T3.5 = 1.750ms to 4.010ms (not *1.5ms to 3.86058ms*).

As of writing it is unclear what causes SysMD to calculate incorrectly, if that is due to SAT/SMT solver which does calculations in behind that it is not supposed to, or if there is an error in the model.

3.2.7 hasA relationship tree-view

hasA relationship tree-view shows the relationship between the elements in a model. This is a valuable tool to check if the model has been described as it was intended and if all inheritances is as expected. Example of ECL product family model can be seen below where it is visible that both *ECL120* and *ECL220* are subclasses of superclass *ECL*, and all have their features correctly adjusted according to the specification.

E.g. The number of *temperatureSensor* instances are adjusted correctly according to the specification.

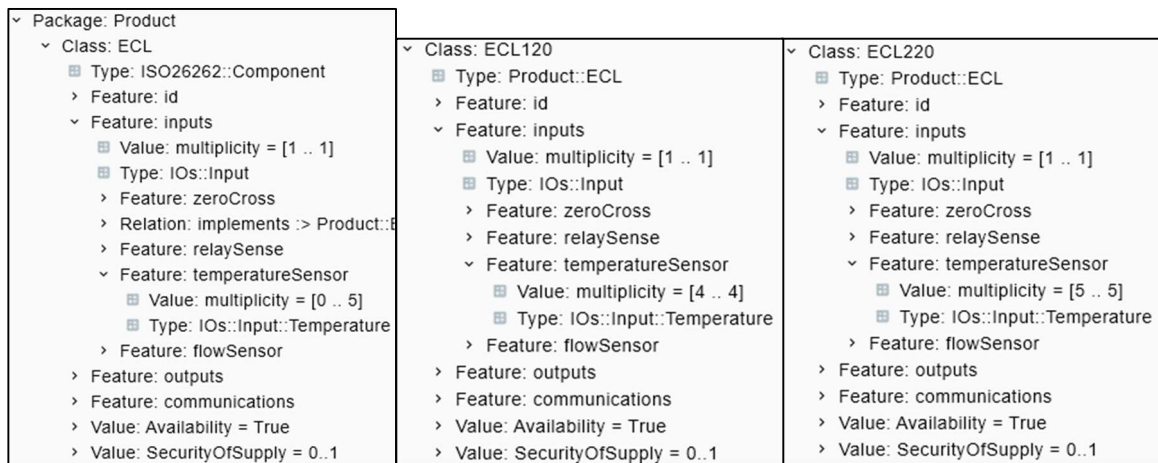


Figure 26 - *hasA* relationship tree-view of package **Product**.

4 Results

This chapter reflects on the results of using SysMD in the ECL project. This chapter will contain a mix of factual information and the authors own opinion about how well SysMD works as a language and how well the tool package performs in a real-world industrial project. The last part of this chapter reflects on the validity and usability around the findings found in this case study.

4.1 Analysis with SAT/SMT solver

The SAT/SMT solver does a general good job of checking for consistency through an entire model whether it is simple consistency checks of refined values from a superclass or calculations of new constraints as can be seen in chapter 3.2.5 and 3.2.6. That said there are still errors and parts where it lacks functionality.

- In chapter 3.2.6 it fails on correctly calculating the values, but it is close at the upper bound. Testing by changing the range limitation to ensure they are not limiting the calculation, the result ends in the right range so it is not completely off. Result can be seen in Table 3 below.

Table 3 - Comparison between correct Modbus T1.5 and T3.5 values and calculated.

	T1.5		T3.5	
	Lower bound	Upper bound	Lower bound	Upper bound
<i>Expected value</i>	0.750ms	1.719ms	1.750ms	4.010ms
<i>Calculated value</i>	0.0ms	2.405ms	0.487ms	3.861ms

- Calculation including either an infinite lower or upper bound causes incorrect results.
- The general lack in SysMD for supporting lists of values instead of only ranges can occasionally result in invalid or unrealistic calculations and results.

4.2 Ability to link documentation with models

The ability to link documentation and models together in SysMD is quite simple but the linking is not that strong. The linking between models and documentation are based on all the information is in the same md file. The distinction between what is documentation and what is model is based on the tag *SysMD* which surrounds all model sections in the md file. The model sections are treated as code sections with the *SysMD* tag in the Markdown language, see example below.

```
## Dependencies
```

```
Following part specifies all dependencies that the following sections depends on.
```

```
```SysMD
```

```
Document imports ScalarValues.
```

```
Document imports Gbo.
```

```
Document imports Types.
```

```
Document imports ZCD.
```

```
Document imports TemperatureSensor.
```

```
Document imports FlowSensor.
```

```
Document imports RS485.
```

```
```
```

Strong linking between documentation and models is best achieved by ensuring each document (individual md file) is only describing a particular group of models thereby ensuring documentation cannot be incorrectly linked inside each document.

4.2.1 Refine documentation with added models

The further refinement of documentation written in SysMD is quite limited compared to what is already available in our existing documentation in Polarion and Word. There is not that much more information in SysMD, however many of requirements which normally is just as text in Word or Polarion has been translated into models in SysMD. This provides the potential of performing a better consistency check, testing, and validation of these requirements.

By specifying the public interfaces of a module in the model, it opens up for the possibility of creating auto-generated code that is fully aligned with documentation. This is a great advantage that provides public interfaces in code which has already been theoretically

validated in SysMD. Example of such an interface specification can be seen in chapter 3.2.4 however it still misses a little information before it can be used for auto-generating the code for the public interface.

4.3 Quality of SysMD Notebook

SysMD and SysMD Notebook is still in its early days which is quite clear from the quality of the tool, which still contains a sizable number of bugs and limitations.

- Development of new feature can cause substantial changes to the syntax of the files.
- Occasionally new development causes tool to crash, functionality stops working, and various minor errors.

In the last ~6 months there has been reported several bugs in the issue tracking system, see Table 4.

Table 4 - Statistics for issue tracking of SysMD in GitLab for the last ~6 months.

| <i>Type reported</i> | <i>Total</i> | <i>By Author</i> | <i>Closed / Fixed</i> |
|------------------------|--------------|------------------|-----------------------|
| <i>Bugs</i> | 44 | 20 | 27 |
| <i>Feature request</i> | 13 | 9 | 2 |
| <i>Question</i> | 5 | 3 | 3 |

There are released versions of SysMD Notebook that is stable enough for daily use, while there are also other versions which are not. Using one of the stable versions, the tool can be used in daily life well knowing the limitations in functionality and known bugs.

4.4 Improvements in SysMD

This section talks about the potential improvements that could be done to either the SysMD language or SysMD Notebook.

4.4.1 Improved error information

The error information is often bad and can be improved a lot to help the user locate the root cause of an error. On many occasions you can guess where the error is as it is most likely in a recently changed model. However sometimes after updating to a newer version of SysMD Notebook you get a new error in existing documentation which had been working perfectly for a long time.

Example of an error like this can be seen on Figure 27. The error appeared after updating to software version 2.9.3 from a 2.8 version. The error was located to origin in the *TemperatureSensor* model, where it was a value with an infinite upper bound limit.

The screenshot shows a SysMD Notebook interface. On the left, there is a 'Product Overview' section with a description of ECL products and a 'Dependencies' section listing eight items: 1 Document imports ScalarValues., 2 Document imports Gbo., 3, 4 Document imports Types., 5 Document imports ZCD., 6 Document imports TemperatureSensor., 7 Document imports FlowSensor., and 8 Document imports RS485. At the bottom of the dependencies list, a red error message is displayed: 'ERROR in Global: During propagation: lateinit property quantity has not been initialized'. On the right side, there is a summary box titled 'Summary: 1 items in agenda.' which contains a red warning icon and the text 'Error during initialization: null'.

Figure 27 – Example of bad error information.

Error information should preferably include following:

- File name
- Element id and line number in specific element.
- Error type id and error text

Error type id should be used for the possibility to get help in FAQ, guides, or communities for guidance and ideas on how to solve a specific error.

4.4.2 Improved relationship syntax

The current relationship syntax is unnecessary complicated and makes the models less readable. The readability in SysMD can be greatly improved by hiding some of the compiler/parser specific needs from the user.

Today a feature relationship is written as following:

```

FeatureB isA Component.
...
FeatureA isA Package.
...
FeatureA hasA
  Component funcA:      [0 .. 2] Component;
  Relation funcA_relation: funcA implements FeatureB;
...

```

This relationship could from a readability point of view be written much clearer by hiding some relationship specific parser requirements. From a user perspective then *funcA_relation* is never used anywhere in SysMD, this is only needed by parser to ensure correct hierarchy.

A more simplified way to write it could be as following:

```

FeatureB isA Component.
...
FeatureA isA Package.
...
FeatureA hasA
  Component funcA: [0 .. 2] Component implements FeatureB;
...

```

Instead of the SysMD parser relies completely on the model specifying the relationship names, it could then automatically create the relationship names itself. To ensure a unique name is always created. The syntax could look something like this:

Relation funcA\$\$\$rel\$\$\$FeatureB: funcA implements FeatureB;

The compiler would be able to take the relationship syntax above and apply that to the model during compilation to get below model which defines a relationship between *funcA* and *FeatureB*:

```
FeatureB isA Component.  
...  
FeatureA isA Package.  
...  
FeatureA hasA  
  Component funcA: [0 .. 2] Component;  
  Relation funcA$$$rel$$$FeatureB: funcA implements FeatureB;  
...
```

This will ensure unique naming of all relationships while hiding this complexity from the user.

4.4.3 Diagrams support

It is great that a lot can be modeled in SysMD and that these models can be visually viewed in the *hasA* tree-view. However this is very much tool dependent and not directly part of documentation. From a documentation point of view it would be nice to have auto-generated diagrams of the component and object type for better visualization, instead of reading a lot of text in a model.

These diagrams could look like the example in Figure 28 of the ECL product overview.

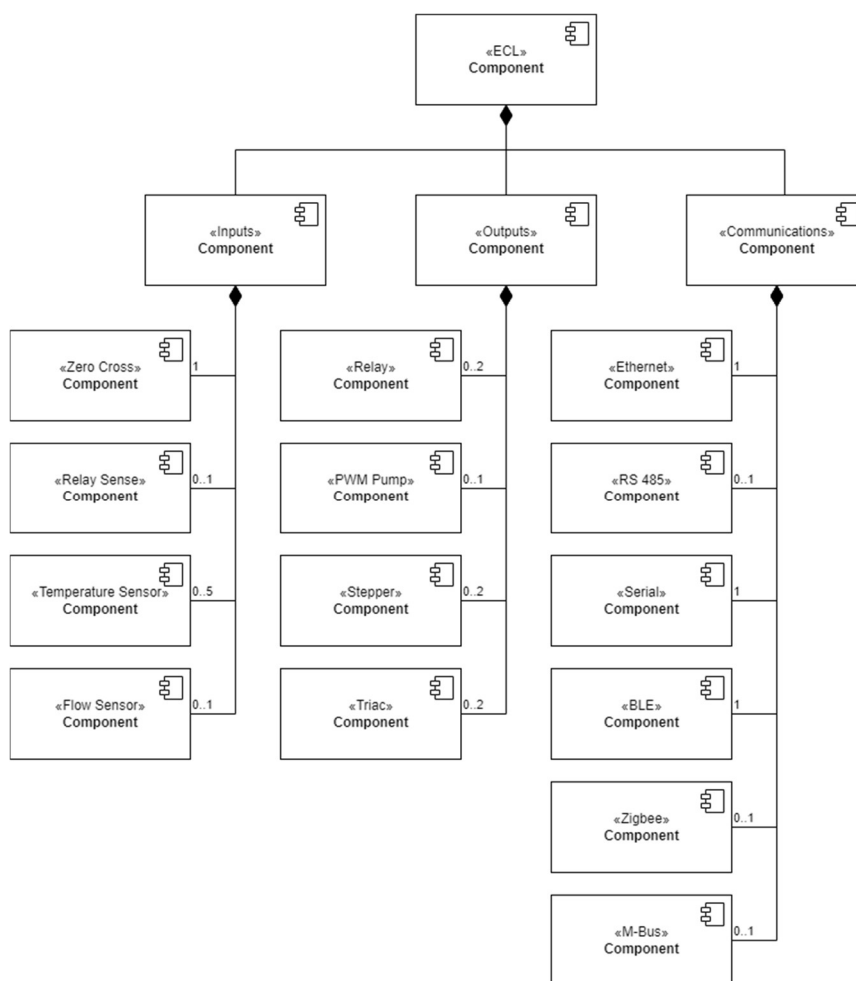


Figure 28 - Component diagram of ECL product overview

A nice addition to this functionality would be the possibility of interactive diagrams where more information could be shown. In the example from Figure 28 it would be nice to interactively overlay the different product variants as they are inheriting from the ECL superclass.

4.4.4 General improvements of SysMD Notebook

SysMD Notebook works okay, but there is a long way to a level where it can compete with some of the best. This leads me to following recommendations:

1. Discontinue SysMD Notebook and replace it with plugin for VS Code or Eclipse.

This might sound as a harsh recommendation, but it provide many other opportunities such as:

- Fully integrated Markdown previewer already available.
- Spell checking available through extension.
- Rest of the recommendation (2. to 4.) already solved.
- Common tool which many users is already familiar with.

The main reason for the suggestion is that the SysMD team can focus their effort on what really matters which is SysMD parser/compiler and does not need to spend time on developing UI/UX interfaces.

2. Visualize if document has unsaved changes.

Currently it is impossible to know if a document has any unsaved changes. If you are not careful and accidently closes SysMD Notebook you would not know if you have lost some data.

3. Notification for unsaved documents.

When closing SysMD Notebook it should check if there are unsaved changes and ask user if they want to save them before closing. Here it would be nice to see what the changes are.

4. Text wrapping

SysMD Notebook should automatically wrap text to fit within the current window size instead of letting the text continue out of view.

4.4.5 MD file linking support

It would be great to have file linking between different MD files. Sometimes it would be great to link to information in another file while there is no direct link on a model level.

4.4.6 Table of content support

In bigger projects it becomes difficult to keep track of all documentation unless there is place which keeps track of everything. In normal documentation this would be the *table of content*, however in SysMD this does not necessarily need to be a standard *table of content* this could also be a visual view of project model from where you can jump to documentation of any parts of the model.

4.4.7 Version control support

Whether a tool is meant for specification, requirement management, or modeling it is dependent on version or change management control as these parts are key factors in normal projects. As SysMD falls under this category it really needs this to ensure it can avoid unwanted changes happening to the documentation and models.

4.5 Validity

Validity of the findings from the case study is somewhat a mix, some parts are valid and will be valid for long time while other parts could already be invalid by tomorrow. It is important to note that SysMD language is very new, and the SysMD Notebook is a tool still under development which means parts might not be completely stable.

The core parts about SysMD, SysMD language, and how those stacks against the findings in the *State of the art* research part can be considered valid. These parts are so much the DNA of the individual tools that it cannot change suddenly.

Anything related to the SysMD Notebook must be carefully assessed, as this is still under development. Therefore things that are valid today could be invalid tomorrow. This highly depends on the development roadmap for the SysMD Notebook.

Another consideration regarding the results found here are that even though a result is valid for the ECL project, it is worth mentioning that it does not mean the result can be transferred to another project as there are many external factors that can affect the results. E.g. If the project needs to consider human safety, then SysMD Notebook will fail miserably in its current state.

The validity of the entire *State of the art* research part cannot be considered scientifically valid for a few reasons:

- Some tools are not only evaluated on pure research but also based on the authors own experience with the tools. This can cause an unwilling bias towards the tool either positive or negative.
- Lack of peer review on rating of the tools.
- Lack of other research papers and reports in the area with the same tools in mind.

Even though it is not scientifically valid, it should still provide an acceptable baseline for use in comparison with SysMD.

4.6 Usability

When talking about usability in SysMD as a language and SysMD Notebook as a tool it is difficult to give an overall rating but looking at it from the perspective of the ECL project, then I can currently not recommend using it.

Splitting the usability rating into a rating for SysMD language and one for SysMD Notebook it gives a slightly different result.

4.6.1 SysMD language

SysMD language has a lot of potential as a language but also has issues.

Pros

- ✓ Easy to read.
- ✓ Simple syntax.
- ✓ Documentation and models integrate very well in regular md files.
- ✓ SI unit support and others.
- ✓ Semantics maps close to SysMLv2.

Cons

- ⊗ Complicated syntax in some cases.
- ⊗ No support for value lists.
- ⊗ Not widely known.

When looking purely at the language itself then I can recommend using it as this provides a good tight coupling between model and documentation, and it is easy to write. With the right tool support it also bridges the gap between model and code due to the possibility of auto-generated code.

Looking a bit broader than the language itself and looking at the parser/compiler then the only supported compiler is still not at a level where that is good enough for a project in the size of the ECL.

4.6.2 SysMD Notebook

Looking sole on the SysMD Notebook then my recommendation is not to use it. It clearly shows that it is still a tool under development and is simply not at a level where it is good enough for daily use in a project like the ECL.

Pros

- ✓ *hasA* and *isA* tree-view.
- ✓ Show of error information from analyze.
- ✓ Syntax coloring.

Cons

- ⊗ Too many errors.
- ⊗ No warning for unsaved files.
- ⊗ Not possible to see if a file has unsaved changes.
- ⊗ No change management or version control.
- ⊗ No text wrapping of long text strings.
- ⊗ Missing generic keyboard shortcuts (bold, italic, ...).
- ⊗ No suggestion or guides for solving errors.
- ⊗ Error information missing file name, error id, and line number.

As seen from the pros/cons list there are many things lacking currently in the tool which makes the tool not ready for primetime in a project like the ECL. Many parts are general things which are available in many other tools, this is also the reason for one of my recommendations to discontinue the tool and create a plugin for VS Code or Eclipse.

Going in that direction will long term release resources from maintaining a tool with user interface and all what comes with that. Instead give more focus to the SysMD parser/compiler which is one of the core parts of SysMD.

5 Conclusion

There exist many tools and languages for specifying requirements and modeling projects. Many of these tools and languages require that the user is a system engineer or modeling expert to properly use them. SysMD is a new language and tool package which is trying to solve this limitation by providing a language which should be easy to use by anyone. This case study has investigated the usability of SysMD and SysMD Notebook in an industrial application.

The comparison of SysMD Notebook with a selected group of existing tools, split between commercial and open-source tools, showed that in many aspects it could not compete with the commercial tools (DOORS, Polarion, and EA). This was most prominent in tool integration, traceability, versioning, and change management. SysMD performed quite well in the SAT/SMT functionality and consistency checking however, was lacking in the other parts of analysis functions where some of the other tools did well in.

The Danfoss product, ECL was used as case in a usability study of SysMD in industrial application. The result from the study showed that SysMD language has a good potential for use in an industrial application and the language was easy to use and provided a good way of mixing both general documentation and models of the project in the same document. On top of that the tool was able to perform analysis for consistency check across the models in a project ensuring requirements did not contradict each other. The simple syntax of SysMD did also have its downside with things that were not possible to document, like list of values.

A specification and modeling language is limited by the tools supporting it, as features become useless without proper tool support. This became very clear with the SysMD language, it has good potential, but its usability was limited by the tool support where SysMD Notebook was as of writing the only tool supporting the language. SysMD Notebook was lacking a lot of features to make it useful, many which are common in standard available editors (e.g. VS Code and Eclipse) as well as more advanced features like version and management control.

During testing of the notebook it was not uncommon that updating to new version fixed errors but gave rise to new ones. The notebook was in general not stable, had many bugs, was not

user friendly which was the reason for several suggested improvements. Result of all this makes SysMD Notebook and thereby SysMD not recommended for use in its current state.

Limitations SysMD language and specially the notebook was still in early stage and during this case study SysMD Notebook was still under development. This is important to consider when evaluating the prospects of SysMD. It does not change the current recommendation, but there could be a good potential for SysMD in future depending on its continued development. The lack of features and amount of bugs in SysMD Notebook has limited the full evaluation of SysMD language as part of the language was not possible to truly validate.

The validity of comparison between SysMD and other tools can be questioned due to lack of research into the same tools and potential bias by author towards some of the tools.

Future Work A full implementation of a complete industrial application into SysMD gives the opportunity to fully test SysMD and evaluate its potential. When doing this full implementation it could benefit from being simultaneous performed in a tool like DOORS, Polarion, or EA for purpose of comparison. This work does not make sense to do before SysMD Notebook has reached a level with less bugs, more stability, and more features available.

This case study has focused on the software part of an industrial application, and therefore there is a lack of investigation into SysMD from a mechanical and hardware point of view.

References

- [1] Danfoss A/S, "District heating," 04 09 2022. [Online]. Available: <https://www.danfoss.com/en/markets/district-energy/dhs/district-heating/#tab-overview>.
- [2] Z. Xinwei, L. Jing, E. Hakki and Z. Chen, "Prioritizing and aggregating interacting requirements for product-service system development," *Expert Systems with Applications*, vol. 185, 2021.
- [3] Wikipedia, "List of requirements engineering tools," 17 12 2022b. [Online]. Available: https://en.wikipedia.org/wiki/List_of_requirements_engineering_tools.
- [4] Siemens Industry Software Inc., "Ovum names Polarion ALM a market leader in new reports," 17 12 2022b. [Online]. Available: <https://polarion.plm.automation.siemens.com/ovum-names-polarion-alm-a-market-leader-in-new-reports>.
- [5] IBM, "Comparison of DOORS and DOORS Next," 10 11 2022b. [Online]. Available: <https://www.ibm.com/docs/en/elm/7.0.0?topic=next-comparison-doors-doors>.
- [6] IBM, "Overview of DOORS," 10 11 2022a. [Online]. Available: <https://www.ibm.com/docs/en/ermd/9.7.2?topic=engineering-requirements-management-doors-overview>.
- [7] IBM, "IBM Engineering Requirements Management DOORS Next," 23 01 2023. [Online]. Available: <https://www.ibm.com/products/requirements-management-doors-next/demos/ai/ai-can-help-you-write-better-quality-requirements>.

- [8] Wikipedia, "Markdown," 06 11 2022a. [Online]. Available: <https://en.wikipedia.org/wiki/Markdown>.
- [9] CommonMark, "CommonMark," 08 11 2022. [Online]. Available: <https://commonmark.org/>.
- [10] Danfoss A/S, "Product Requirement Specification (Redacted)," 2022.
- [11] Agile Business Consortium Limited, "MoSCoW Prioritisation," 17 12 2022. [Online]. Available: <https://www.agilebusiness.org/dsdm-project-framework/moscow-prioritisation.html>.
- [12] N. B. Nielsen, Interviewee, *User experience with Polarion*. [Interview]. 06 10 2022.
- [13] Siemens Industry Software Inc., "Polarion - Requirements," 06 10 2022a. [Online]. Available: <https://polarion.plm.automation.siemens.com/products/polarion-requirements>.
- [14] Sparx Systems Pty Ltd., "Enterprise Architect," 06 01 2023. [Online]. Available: <https://sparxsystems.com/>.
- [15] Sparx Systems Pty Ltd., "Enterprise Architect - Product Overview," 06 01 2023. [Online]. Available: <https://sparxsystems.com/products/ea/index.html>.
- [16] Wikipedia, "Enterprise Architect (software)," 06 01 2023. [Online]. Available: [https://en.wikipedia.org/wiki/Enterprise_Architect_\(software\)](https://en.wikipedia.org/wiki/Enterprise_Architect_(software)).
- [17] Project Jupyter, "Jupyter," 22 09 2022a. [Online]. Available: <https://jupyter.org/>.

- [18] Project Jupyter, "The Jupyter Notebook," 22 09 2022b. [Online]. Available: <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>.
- [19] Jupyter Contrib Team, "Unofficial Jupyter Notebook Extensions," 06 01 2023. [Online]. Available: <https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/>.
- [20] R. Lacok, "Data Science Notebooks - Version control in Jupyter notebooks," 05 01 2023. [Online]. Available: <https://datasciencenotebook.org/jupyter-version-control>.
- [21] SysML v2 Submission Team (SST), "SysML v2 Release," 01 08 2022. [Online]. Available: <https://github.com/Systems-Modeling/SysML-v2-Release>.
- [22] C. Grimm, "SysMD @ OMG," Chair of Cyber-Physical Systems, TU Kaiserslautern, Kaiserslautern, 2022.
- [23] M. Hoffmann, N. Kühn, M. Weber and M. Bittner, "Requirements for Requirements Management Tools," in *12th IEEE International Requirements Engineering Conference (RE'04)*, 2004.
- [24] M. F. Santillán, "Criteria for the evaluation of requirements management tools supporting distributed software product line engineering and management," University of Jyväskylä, Jyväskylä, 2015.
- [25] S. Dalecke, K. A. Rafique, A. Ratzke, C. Grimm and J. Koch, "SysMD: Towards "Inclusive" System Engineering," Chair of Cyber-Physical Systems, TU Kaiserslautern, Kaiserslautern, 2022.
- [26] C. Grimm, S. Post, S. Dalecke, J. Koch, A. Ratzke and F. Wawrizik, "SysMD Language Reference Manual," Chair of Cyber-Physical Systems, TU Kaiserslautern, 2022.

- [27] Modbus.org, "MODBUS over serial line specification and implementation guide V1.02," 2006.
- [28] O. O. Oduko, "Comparison of Requirement Management Software," University of Alberta, Alberta, 2021.
- [29] T. Virtanen, "Comparison of Requirements Management Solutions for Medical Device Development," Metropolia University of Applied Sciences, Helsinki, 2020.
- [30] Software Education, "Control changes to the requirements with Rational DOORS Next Generation," 11 10 2022. [Online]. Available:
<https://www.youtube.com/watch?v=G65e5yaxlAI&list=PLZGO0qYNSD4V5mE7UZJhuRmRZtmjrjwwK&index=4>.

Appendix

Appendix: A SysMD project files

This appendix contains all the md files from SysMD Notebook project for ECL.

Appendix: A.1 ECL.md

Table of content

TODO Full table of content across all the different md files.

[[TOC]]

- TOC

{:toc}

{(TOC)}

[TOC]

Would look something like this if had table of content support.

1. ECL x20 Software Platform
 1. Background
2. Product Overview
 1. Dependencies
 2. Product description for base ECL product family
3. Global Types
 1. Dependencies
 2. Function arguments
4. Temperature Sensor Module
 1. Dependencies
 2. Requirements
 3. API
5. Flow Sensor Module
 1. Dependencies
 2. Requirements
 3. API
6. ...

Background

The ECL Platform is a SW/HW platform which support the new generation of ECLs currently consisting of:

- ECL 120 More details
- ECL 220 More details

This document describes the overall SW requirements and SW architecture of the ECL platform. The document is split into different parts:

- Platform overview.
- Products covered by this platform (variants).
- Component details.

Some more text here

Appendix: A.2 Types.md

Global Types

This file contains general types

Dependencies

Following specifies all dependencies which this module depends on:

```
Document imports ScalarValues.  
Document imports Gbo.
```

Function arguments

```
Types isA Component.  
Types defines  
  Arg isA Function;  
  HwType isA Function.
```

```
Types::Arg defines  
  Input      isA Function;  
  Output     isA Function;  
  Callback   isA Function.
```

```
Types::HwType defines  
  ECL      isA Component;  
  ECL120 isA ECL;  
  ECL220 isA ECL.  
  
Types::HwType::ECL      hasA Value type: Real(120 .. 220).  
Types::HwType::ECL120 hasA Value type: Real(120) = 120.0.  
Types::HwType::ECL220 hasA Value type: Real(220) = 220.0.
```

Appendix: A.3 Product.md

Product Overview

This part describes the overview of the ECL products. Which products exist in the family, names of products and the main differences of the products.

Dependencies

Following part specifies all dependencies that the following sections depends on.

```
Document imports ScalarValues.
Document imports Gbo.

Document imports Types.
Document imports ZCD.
Document imports TemperatureSensor.
Document imports FlowSensor.
Document imports RS485.
```

Product type definitions

```
Product isA Package.
Product defines
  ECL    isA Component;
  ECL120 isA ECL;
  ECL220 isA ECL.

Product::ECL    hasA Component id: Types::HwType::ECL.
Product::ECL120 hasA Component id: Types::HwType::ECL120.
Product::ECL220 hasA Component id: Types::HwType::ECL220.
```

Connections

Following describes the available type of inputs, outputs and communications for the ECL product family. This gives the complete overview of different types but does not specify what each product includes.

```
IOs isA Component.
IOs defines
  IO      isA Component;
  Input   isA IO;
  Output  isA IO.
```

List of available inputs.

```
IOs::Input defines
  Temperature isA Component;
  RelaySense  isA Component;
  ZeroCross   isA Component;
  Flow        isA Component.
```

List of available outputs.

```
IOs::Output defines
  Relay    isA Output;
  Triac    isA Output;
  Stepper  isA Output;
  PumpPWM  isA Output.
```

List of available communication interfaces.

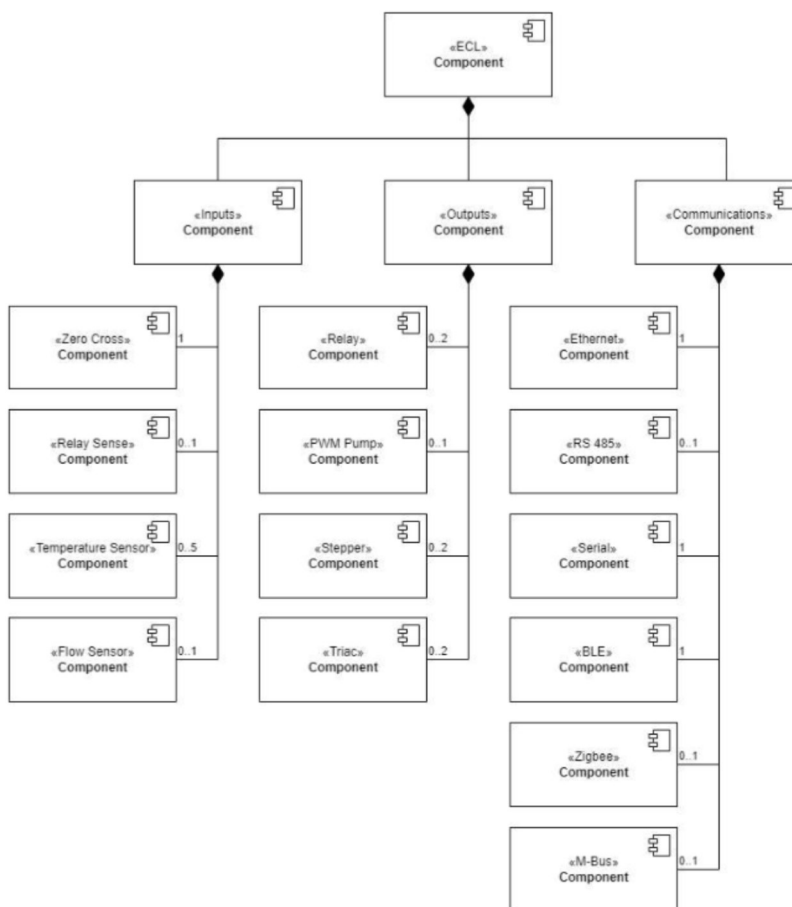
```

Comms isA Component.
Comms defines
  Comm isA Component;
  Ethernet isA Comm;
  RS485 isA Comm;
  MBus isA Comm;
  Zigbee isA Comm;
  Serial isA Comm;
  BLE isA Comm.

```

Product description for base ECL product family

Following describes the basic features of any ECL. It includes which inputs, outputs, communications and so on that it supports. This is features that is not specific for a single product type. Product type specific features is describe separate sections.



```

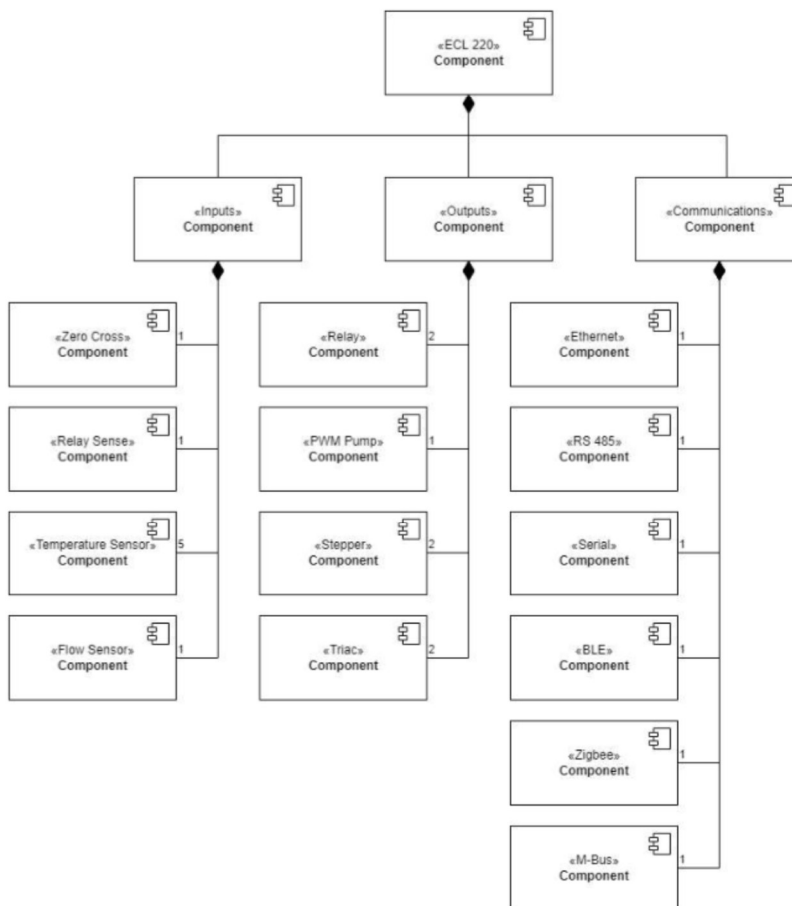
Product::ECL hasA
  Component inputs:      IOs::Input;
  Component outputs:     IOs::Output;
  Component communications: Comms.

Product::ECL::inputs hasA
  Component zeroCross:      [1 .. 1] IOs::Input::ZeroCross;
  Relation zeroCross_rel:   zeroCross implements ZCD;
  Component relaySense:     [0 .. 1] IOs::Input::RelaySense;
  Component temperatureSensor: [0 .. 5] IOs::Input::Temperature;
  Relation temperatureSensor_rel: temperatureSensor implements
TemperatureSensor;
  Component flowSensor:     [0 .. 1] IOs::Input::Flow;
  Relation flowSensor_rel:  flowSensor implements FlowSensor.

Product::ECL::outputs hasA
  Component relay:          [0 .. 2] IOs::Output::Relay;
  Component pumpPwm:       [0 .. 1] IOs::Output::PumpPWM;
  Component triac:         [0 .. 2] IOs::Output::Triac;
  Component stepper:       [0 .. 2] IOs::Output::Stepper.

Product::ECL::communications hasA
  Component ethernet:      [1 .. 1] Comms::Ethernet;
  Component rs485:         [0 .. 1] Comms::RS485;
  Relation rs485_rel:      rs485 implements RS485;
  Component serial:        [1 .. 1] Comms::Serial;
  Component ble:           [1 .. 1] Comms::BLE;
  Component zigbee:        [0 .. 1] Comms::Zigbee;
  Component mbus:          [0 .. 1] Comms::MBus.
    
```

Variance description for ECL220 compared to base functionality of an ECL.



{width=762 height=861}

```

Product::ECL220::inputs hasA
  Component zeroCross:      [1 .. 1] IOs::Input::ZeroCross;
  Component relaySense:     [1 .. 1] IOs::Input::RelaySense;
  Component temperatureSensor: [5 .. 5] IOs::Input::Temperature;
  Component flowSensor:     [1 .. 1] IOs::Input::Flow.

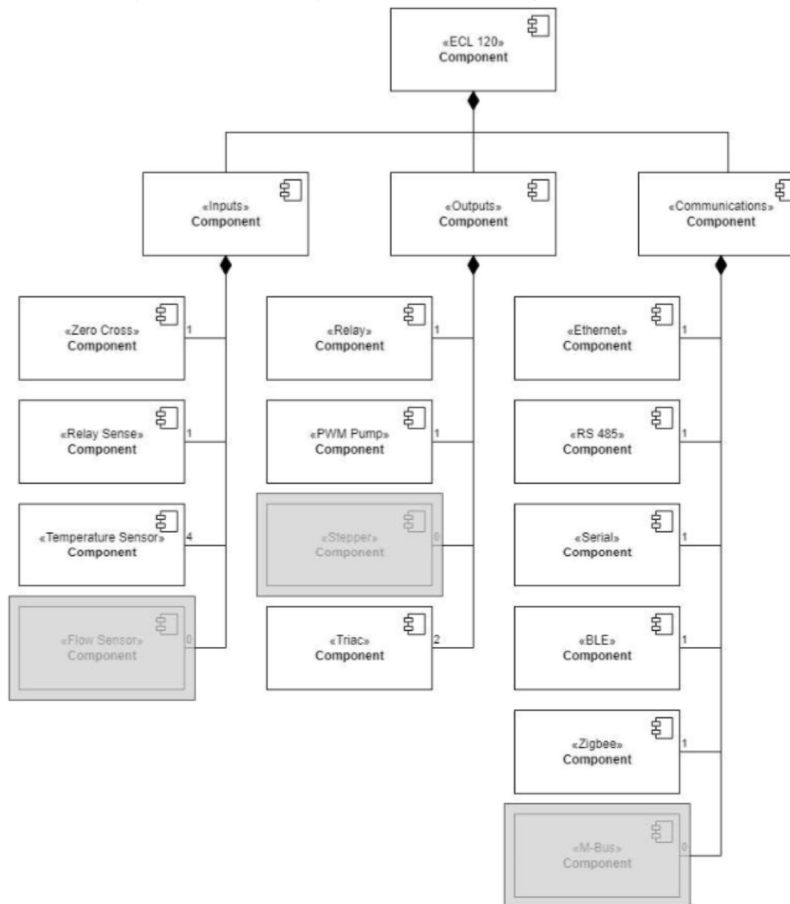
Product::ECL220::inputs::temperatureSensor::TemperatureSensor::Req::SampleRate
hasA
  Value rate: Real(5 .. 10)[Hz].

Product::ECL220::outputs hasA
  Component relay:          [2 .. 2] IOs::Output::Relay;
  Component pumpPwm:       [1 .. 1] IOs::Output::PumpPWM;
  Component triac:         [2 .. 2] IOs::Output::Triac;
  Component stepper:       [2 .. 2] IOs::Output::Stepper.

Product::ECL220::communications hasA
  Component ethernet:      [1 .. 1] Comms::Ethernet;
  Component rs485:         [1 .. 1] Comms::RS485;
  Component serial:        [1 .. 1] Comms::Serial;
  Component ble:           [1 .. 1] Comms::BLE;
  Component zigbee:        [1 .. 1] Comms::Zigbee;
  Component mbus:          [1 .. 1] Comms::MBus.

```

Variance description for ECL120 compared to base functionality of an ECL.



(width=762 height=861)

```
Product::ECL120::inputs hasA
  Component zeroCross:      [1 .. 1] IOs::Input::ZeroCross;
  Component relaySense:     [1 .. 1] IOs::Input::RelaySense;
  Component temperatureSensor: [4 .. 4] IOs::Input::Temperature;
  Component flowSensor:     [0 .. 0] IOs::Input::Flow.

Product::ECL120::outputs hasA
  Component relay:          [1 .. 1] IOs::Output::Relay;
  Component pumpPwm:       [1 .. 1] IOs::Output::PumpPWM;
  Component triac:         [2 .. 2] IOs::Output::Triac;
  Component stepper:       [0 .. 0] IOs::Output::Stepper.

Product::ECL120::communications hasA
  Component ethernet:      [1 .. 1] Comms::Ethernet;
  Component rs485:        [1 .. 1] Comms::RS485;
  Component serial:       [1 .. 1] Comms::Serial;
  Component ble:          [1 .. 1] Comms::BLE;
  Component zigbee:       [1 .. 1] Comms::Zigbee;
  Component mbus:         [0 .. 0] Comms::MBus.
```

New thing here ... Not sure what yet

```
Mains isA Package.
Mains defines
  Rating isA Component.

Mains::Rating hasA
  voltage:  Real(220 .. 240)[V];
  freq:    Real(50 .. 60)[Hz].
```

...

Appendix: A.4 TemplateModule.md

<module name> Module

This module main purpose is to <module description>.

Dependencies

Following specifies all dependencies which this module depends on:

```
Document imports ScalarValues.
Document imports Gbo.

Document imports <module dependency>.
```

Specification

Public specification for this module.

```
<module name> isA Component
<module name> defines
  Req    isA Requirement;
  API    isA Function;
  TestAPI isA Function.
```

Requirements

Following section describes the specific requirements there is for the <module name> component.

```
<module name>::Req defines
  ReqA  isA Requirement;
  ReqB  isA Requirement.

<module name>::Req::ReqA hasA
  Value limitA:  Integer(0 .. 5);
  Value limitB:  Real(0 .. 5).
```

Public API

Following section describes the public API available by the <module name> component.

```
<module name>::API defines
  FunctionA  isA Function;
  FunctionB  isA Function.

<module name>::API::FunctionA hasA
  Value inArgA:  Real(0 .. 5).
```

Test API

Following section describes the test API available by the <module name> component.

```
<module name>::TestAPI defines
  FunctionA  isA Function;
  FunctionB  isA Function.

<module name>::TestAPI::FunctionA hasA
  Value inArgA:  Real(0 .. 5).
```

Appendix: A.5 TemperatureSensor.md

Temperature Sensor Module

ECL 120 and ECL 220 has each temperature sensor inputs. These are used to measure relevant temperatures for the application whether this being temperature sensors internally in the flat station or being room/outdoor temperature sensors.

These temperature sensors must be measured at regular interval and with good repeatability to ensure the usability of the MBD regulation applications.

Dependencies

Following specifies all dependencies which this module depends on:

```
Document imports ScalarValues.
Document imports Gbo.

Document imports Types.
```

Specification

Public specification for this module.

```
TemperatureSensor isA Component.
TemperatureSensor defines
  Req isA Requirement;
  API isA Function;
  TestAPI isA Function.
```

Requirements

Following section describes the specific requirements there is for the temperature sensor component.

```
TemperatureSensor::Req defines
  Range isA Requirement;
  AccurateRange isA Range;
  SampleRate isA Requirement;
  SupportedSensor isA Requirement;
  NoiseImmunity isA Requirement.

TemperatureSensor::Req::Range hasA
  Value range: Real(-64 .. 192) [°C];
  Value accuracy: Real(-5 .. 5) [K];
  Value resolution: Real(0 .. 0.1) [K];
  Value repeatability: Real(-1 .. 1) [K].

TemperatureSensor::Req::AccurateRange hasA
  Value range: Real(-10 .. 95) [°C];
  Value accuracy: Real(-2 .. 2) [K];
  Value resolution: Real(0 .. 0.01) [K];
  Value repeatability: Real(-0.1 .. 0.1)[K].

TemperatureSensor::Req::SampleRate hasA
  Value rate: Real(2 .. 10)[Hz].

TemperatureSensor::Req::SupportedSensor hasA
  Value ptType: Real(1000);
  Value maxNumOfSensor: Integer(0 .. 5).

TemperatureSensor::Req::NoiseImmunity hasA
  Value freq: Real(50 .. 60)[Hz].
```


API

Following section describes the public API available by the temperature sensor component.

```

TemperatureSensor::API defines
  TempSensor_IsSupported           isA Function;
  TempSensor_GetTemperature        isA Function;
  TempSensor_SetRTD                isA Function;
  TempSensor_ApplyCalibrationValue isA Function;
  TempSensor_Handler               isA Function;
  TempSensor_Init                  isA Function.

TemperatureSensor::API::TempSensor_IsSupported hasA
  Value inSensorId: Integer =
TemperatureSensor::Req::SupportedSensor::maxNumOfSensor.

TemperatureSensor::API::TempSensor_GetTemperature hasA
  Value inSensorId: Integer =
TemperatureSensor::Req::SupportedSensor::maxNumOfSensor;
  Value outStatus: Integer(0 .. 3);
  Value outValue: Real = TemperatureSensor::Req::Range::range.

TemperatureSensor::API::TempSensor_SetRTD hasA
  Value inSensorId: Integer =
TemperatureSensor::Req::SupportedSensor::maxNumOfSensor;
  Value inR0: Real(0 .. 2000);
  Value inA: Real(-1 .. 1);
  Value inB: Real(-1 .. 1);
  Value inC: Real(-10 .. 10).

TemperatureSensor::API::TempSensor_ApplyCalibrationValue hasA
  Value inSensorId: Integer =
TemperatureSensor::Req::SupportedSensor::maxNumOfSensor;
  Value inValue: Real(-200 .. 200).

TemperatureSensor::API::TempSensor_Init hasA
  Function inCallback: Types::Arg::Callback;
  Value inId: Real(0 .. 2E32);
  Component inHwType: Types::HwType::ECL.

```

```

TemperatureSensor::TestAPI defines
  TempSensor_Test_SetSimulatedResistance isA Function;
  TempSensor_Test_GetRawResistance      isA Function;
  TempSensor_Test_DisableSimulatedResistance isA Function.

```

Appendix: A.6 FlowSensor.md

Flow Sensor Module

This module main purpose is to handle flow sensor input.

Dependencies

Following specifies all dependencies which this module depends on:

```
Document imports ScalarValues.  
Document imports Gbo.
```

Specification

Public specification for this module.

```
FlowSensor isA Component.  
FlowSensor defines  
  Req      isA Requirement;  
  API      isA Function;  
  TestAPI isA Function.
```

Requirements

Following section describes the specific requirements there is for the flow sensor component.

```
FlowSensor::Req defines  
  Range isA Requirement.  
  
FlowSensor::Req::Range hasA  
  Value frequencyRange: Real(0 .. 300) [Hz];  
  Value resolution:     Real(0 .. 0.01) [Hz];  
  Value accuracy:       Real(-1 .. 1) [Hz].
```

Public API

Following section describes the public API available by the flow sensor component.

```
FlowSensor::API defines  
  FlowSensor_Init      isA Function;  
  FlowSensor_GetFrequency isA Function.  
  
FlowSensor::API::FlowSensor_GetFrequency hasA  
  Value retFreq: Real(0 .. 500) [Hz].
```

Test API

Following section describes the test API available by the flow sensor component.

```
FlowSensor::TestAPI defines
  FlowSensor_SetSimulatedFrequency    isA Function;
  FlowSensor_SetSimulatedRawData      isA Function;
  FlowSensor_DisableSimulatedInput    isA Function;
  FlowSensor_GetMeasuredRawData       isA Function.

FlowSensor::TestAPI::FlowSensor_SetSimulatedFrequency hasA
  Value inFreq:      Real(0 .. 300) [Hz].

FlowSensor::TestAPI::FlowSensor_SetSimulatedRawData hasA
  Value inFirstRisingTick: Real(0 .. *) [ms];
  Value inLastRisingTick:  Real(0 .. *) [ms];
  Value inNoOfTriggers:    Integer(0 .. *);
  Value inCurrentTick:     Real(0 .. *) [ms];
  Value retStatus:        Integer(0 .. 1). // 0 = Invalid input, 1 =
Success

FlowSensor::TestAPI::FlowSensor_GetMeasuredRawData hasA
  Value outFirstRisingTick: Real(0 .. *) [ms];
  Value outLastRisingTick:  Real(0 .. *) [ms];
  Value outNoOfTriggers:    Integer(0 .. *);
  Value outCurrentTick:     Real(0 .. *) [ms].
```

Appendix: A.7 RS485.md

RS485 Module

This module main purpose is to provide RS485 communication to the ECL. This module is the basis for supporting communication protocols Modbus and BACnet.

Dependencies

Following specifies all dependencies which this module depends on:

```
Document imports ScalarValues.
Document imports Gbo.
Document imports SI.
Document uses SI.
```

Specification

Public specification for this module.

```
RS485 isA Component.
RS485 defines
  Req    isA Requirement;
  API    isA Function;
  TestAPI isA Function.
```

Requirements

Following section describes the specific requirements there is for the RS485 component.

```
RS485::Req defines
  Interface  isA Requirement;
  Protocol   isA Requirement;
  Modbus     isA Requirement.

RS485::Req::Interface hasA
  Value baudrate:      Integer(9600 .. 115200);    // 9600, 19200, 38400,
57600, 76800, 115200
  Value parity:        Integer(0 .. 2);           // 0 = None, 1 = Even, 2 =
Odd
  Value bias:          Boolean.                   // False = Off, True = On

RS485::Req::Protocol hasA
  Value supportModbus: Boolean(true);             // True = Supporting
Modbus
  Value supportBACnet: Boolean.                   // True = Supporting
BACnet

RS485::Req::Modbus imports SI.
RS485::Req::Modbus hasA
  Value charBitSize:   Integer(11 .. 11);
  Value t1_5Timeout:  Time(500.0e-3 .. 2.0) [ms]
    = ITE(RS485::Req::Interface::baudrate > 19200, 750e-3[ms], 1.5 /
Real(RS485::Req::Interface::baudrate / charBitSize)); // Inter character timeout
on bus. Above 19200 = 750us
  Value t3_5Timeout:  Time(1.50 .. 5.0) [ms]
    = ITE(RS485::Req::Interface::baudrate > 19200, 1.750[ms], 3.5 /
Real(RS485::Req::Interface::baudrate / charBitSize)). // Inter frame timeout on
bus. Above 19200 = 1.750ms
```

Public API

Following section describes the public API available by the RS485 component.

```
RS485::API defines
  vMBPortReceiveBlock isA Function;
  vMBPortSendBlock   isA Function;
  xMBPortSerialInit  isA Function.

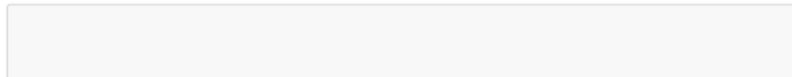
RS485::API::vMBPortReceiveBlock hasA
  Value outBuf:      Real(0 .. *);
  Value inMaxSize:  Integer(0 .. 256).

RS485::API::vMBPortSendBlock hasA
  Value inBuf:      Real(0 .. *);
  Value inSize:    Integer(0 .. 256).

RS485::API::xMBPortSerialInit hasA
  Value inPort:     Integer(0 .. *); // USART port number
  Value inBaudRate: Integer = RS485::Req::Interface::baudrate;
  Value inDataBits: Integer(0 .. 9);
  Value inParity:   Integer = RS485::Req::Interface::parity.
```

Test API

Following section describes the test API available by the RS485 component.



Appendix: A.8 ZCD.md

Zero Cross Detection Module

This module main purpose is to detect AC mains frequency (230V @ 50/60Hz) of the product.

Dependencies

Following specifies all dependencies which this module depends on:

```
Document imports ScalarValues.
Document imports Gbo.
```

Specification

Public specification for this module.

```
ZCD isA Component.
ZCD defines
  Req    isA Requirement;
  API    isA Function;
  TestAPI isA Function.
```

Requirements

Following section describes the specific requirements there is for the zero cross detection component.

```
ZCD::Req defines
  FreqRangeLimit isA Requirement;
  FreqRangeDetect isA FreqRangeLimit.

ZCD::Req::FreqRangeDetect hasA
  Value range:      Real(50 .. 60) [Hz];
  Value accuracy:   Real(-1 .. 1) [Hz];
  Value resolution: Real(0 .. 0.5) [Hz];
  Value voltage:    Real(220 .. 240)[V].

ZCD::Req::FreqRangeLimit hasA
  Value range:      Real(40 .. 70) [Hz];
  Value accuracy:   Real(-1 .. 1) [Hz];
  Value resolution: Real(0 .. 0.5) [Hz].
```

Public API

Following section describes the public API available by the zero cross detection component.

```
ZCD::API defines
  ZCD_GetFreqDetect isA Function;
  ZCD_GetFreqValue isA Function;
  ZCD_AwaitZeroCross isA Function.

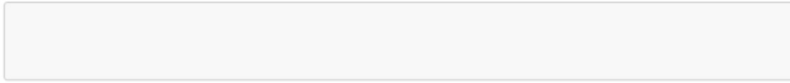
ZCD::API::ZCD_GetFreqDetect hasA
  Value retFreq:      Real(0 .. 60) [Hz].    // Return values: 0 = Not detected,
  50/60 = Detected frequency [Hz]

ZCD::API::ZCD_GetFreqValue hasA
  Value outCurFreq:  Real(0 .. 70) [Hz];
  Value outMinFreq:  Real(0 .. 70) [Hz];
  Value outMaxFreq:  Real(0 .. 70) [Hz].

ZCD::API::ZCD_AwaitZeroCross hasA
  Value inTimeout:   Real(0 .. 1) [s];      // Max wait time for zero cross.
  Value retStatus:   Boolean.              // False = Timeout, True = Success
```

Test API

Following section describes the test API available by the zero cross detection component.



Appendix: B Tools comparison evaluation

This appendix contains the raw evaluation used to rate the different tools. The evaluation is based on rating from 1 to 5 where 5 is equal very good. Each category is rated based on a set of questions to answer where each question has a weight of the combined rating.

Appendix: B.1 Category Results

| | DOORS | Excel | Polarion | Markdown | Enterprise Architect | Jupyter Notebook | SysMD Notebook |
|---|----------|-------|----------|----------|----------------------|------------------|----------------|
| Criteria categories | Document | | | | Model/Mixed | Notebook | |
| Open-source / Open-standards | 2,8 | 3,25 | 2,8 | 4,8 | 4,3 | 4,95 | 4,8 |
| Modeling & Testing | 1,75 | 1 | 1,75 | 1 | 4,75 | 1,55 | 3,35 |
| Analysis functions (SAT/SMT) | 3,4 | 1,75 | 2,5 | 1 | 3,7 | 1 | 2,3 |
| Traceability | 4,4 | 2,9 | 4,4 | 2,6 | 4,4 | 1 | 2,7 |
| Version & Change control | 4,65 | 2,75 | 4,65 | 1,2 | 4,65 | 1,2 | 1,2 |
| Tool integration | 3,9 | 3 | 4,5 | 1,4 | 3,9 | 3,25 | 1 |
| Import | 3,5 | 2 | 3,5 | 1 | 4 | 1 | 1 |
| Formatting, Multimedia & External files | 3,6 | 3,4 | 4 | 3 | 4 | 2,4 | 3 |
| Document Proofing & Generation | 4,45 | 2,55 | 4,2 | 1 | 4,2 | 1,5 | 1 |
| Collaboration | 2,8 | 3,7 | 3,5 | 1,5 | 4,8 | 2 | 1,5 |
| User interface & Usability | 4 | 2,5 | 4 | 1,8 | 4 | 2,8 | 3 |

Conversion between rating and labels:

[4.5 .. 5.0] = ▲▲ [3.5 .. 4.5] = ▲ [2.5 .. 3.5] = ● [1.5 .. 2.5] = ▼ [1.0 .. 1.5] = ▼▼

| | DOORS | Excel | Polarion | Markdown | Enterprise Architect | Jupyter Notebook | SysMD Notebook |
|---|----------|-------|----------|----------|----------------------|------------------|----------------|
| Criteria categories | Document | | | | Model/Mixed | Notebook | |
| Open-source / Open-standards | ● | ● | ● | ▲▲ | ▲ | ▲▲ | ▲▲ |
| Modeling & Testing | ▼ | ▼▼ | ▼ | ▼▼ | ▲▲ | ▼ | ● |
| Analysis functions (SAT/SMT) | ● | ▼ | ● | ▼▼ | ▲ | ▼▼ | ▼ |
| Traceability | ▲ | ● | ▲ | ● | ▲ | ▼▼ | ● |
| Version & Change control | ▲▲ | ● | ▲▲ | ▼▼ | ▲▲ | ▼▼ | ▼▼ |
| Tool integration | ▲ | ● | ▲▲ | ▼▼ | ▲ | ● | ▼▼ |
| Import | ▲ | ▼ | ▲ | ▼▼ | ▲ | ▼▼ | ▼▼ |
| Formatting, Multimedia & External files | ▲ | ● | ▲ | ● | ▲ | ▼ | ● |
| Document Proofing & Generation | ▲ | ● | ▲ | ▼▼ | ▲ | ▼ | ▼▼ |
| Collaboration | ● | ▲ | ▲ | ▼ | ▲▲ | ▼ | ▼ |
| User interface & Usability | ▲ | ● | ▲ | ▼ | ▲ | ● | ● |

Appendix: B.2 Sub-category Evaluation

| Rating: 1-5 | | Rating | | | | | | | | | Weighted | | | | | |
|--|--|--------|-------|-------|----------|----------|----------------------|------------------|----------------|-------|----------|----------|----------|----------------------|------------------|----------------|
| Criteria | Sub criteria | Weight | DOORS | Excel | Polarion | Markdown | Enterprise Architect | Jupyter Notebook | SysMD Notebook | DOORS | Excel | Polarion | Markdown | Enterprise Architect | Jupyter Notebook | SysMD Notebook |
| Open-source / Open-standards | Is tool open-source? | 15% | 1 | 1 | 1 | 5 | 1 | 5 | 5 | 0,2 | 0,2 | 0,2 | 0,8 | 0,2 | 0,8 | 0,8 |
| Open-source / Open-standards | Is tool using open-standards? | 50% | 2 | 3 | 2 | 5 | 5 | 5 | 5 | 1 | 1,5 | 1 | 2,5 | 2,5 | 2,5 | 2,5 |
| Open-source / Open-standards | Is tool support exporting data in open-standards formats? | 30% | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 1,5 | 1,5 | 1,5 | 1,5 | 1,5 | 1,5 | 1,5 |
| Open-source / Open-standards | Is tool support open-standard APIs? | 5% | 3 | 2 | 3 | 1 | 3 | 4 | 1 | 0,2 | 0,1 | 0,2 | 0,1 | 0,2 | 0,2 | 0,1 |
| Modeling & Testing | Is tool using standard modeling languages (SysML, UML)? | 20% | 1 | 1 | 1 | 1 | 5 | 1 | 2 | 0,2 | 0,2 | 0,2 | 0,2 | 1 | 0,2 | 0,4 |
| Modeling & Testing | Can requirements be modeled (visual or textual)? | 30% | 1 | 1 | 1 | 1 | 5 | 2 | 4 | 0,3 | 0,3 | 0,3 | 0,3 | 1,5 | 0,6 | 1,2 |
| Modeling & Testing | Can model be visual represented? | 25% | 2 | 1 | 2 | 1 | 5 | 2 | 4 | 0,5 | 0,3 | 0,5 | 0,3 | 1,3 | 0,5 | 1 |
| Modeling & Testing | Is tool able to verify and validate individual model/requirements? | 25% | 3 | 1 | 3 | 1 | 4 | 1 | 3 | 0,8 | 0,3 | 0,8 | 0,3 | 1 | 0,3 | 0,8 |
| Analysis functions (SAT/SMT) | Is tool supporting SAT/SMT or other consistency and constraint checkers? | 40% | 2 | 1 | 1 | 1 | 4 | 1 | 3 | 0,8 | 0,4 | 0,4 | 0,4 | 1,6 | 0,4 | 1,2 |
| Analysis functions (SAT/SMT) | Is tool supporting scanning of text for unsuitable, inexact language and terminology? | 25% | 4 | 2 | 2 | 1 | 2 | 1 | 1 | 1 | 0,5 | 0,5 | 0,3 | 0,5 | 0,3 | 0,3 |
| Analysis functions (SAT/SMT) | Is tool supporting analysis for missing links and gaps in traceability? | 20% | 5 | 2 | 5 | 1 | 5 | 1 | 2 | 1 | 0,4 | 1 | 0,2 | 1 | 0,2 | 0,4 |
| Analysis functions (SAT/SMT) | Is error reporting to user useful? | 15% | 4 | 3 | 4 | 1 | 4 | 1 | 3 | 0,6 | 0,5 | 0,6 | 0,2 | 0,6 | 0,2 | 0,5 |
| Traceability | Is it easy to link requirements? | 30% | 4 | 2 | 4 | 1 | 4 | 1 | 3 | 1,2 | 0,6 | 1,2 | 0,3 | 1,2 | 0,3 | 0,9 |
| Traceability | How flexible is the linking (attributes, objects, and linking direction)? | 40% | 5 | 5 | 5 | 5 | 5 | 1 | 3 | 2 | 2 | 2 | 2 | 2 | 0,4 | 1,2 |
| Traceability | Is there a good overview and navigation for tracing links? | 30% | 4 | 1 | 4 | 1 | 4 | 1 | 2 | 1,2 | 0,3 | 1,2 | 0,3 | 1,2 | 0,3 | 0,6 |
| Version & Change control | Is it possible to track any changes happened and who did the change? | 25% | 5 | 3 | 5 | 1 | 5 | 1 | 1 | 1,3 | 0,8 | 1,3 | 0,3 | 1,3 | 0,3 | 0,3 |
| Version & Change control | Is it possible to track changes between different released versions? | 15% | 4 | 3 | 4 | 1 | 4 | 1 | 1 | 0,6 | 0,5 | 0,6 | 0,2 | 0,6 | 0,2 | 0,2 |
| Version & Change control | Is the tool supporting version control and user friendly? | 15% | 5 | 3 | 5 | 1 | 5 | 1 | 1 | 0,8 | 0,5 | 0,8 | 0,2 | 0,8 | 0,2 | 0,2 |
| Version & Change control | Is it possible to use GIT/SVN indirectly in a useful way? | 5% | 1 | 2 | 1 | 5 | 1 | 5 | 5 | 0,1 | 0,1 | 0,1 | 0,3 | 0,1 | 0,3 | 0,3 |
| Version & Change control | Is it possible to review changes before they are implemented? | 20% | 5 | 3 | 5 | 1 | 5 | 1 | 1 | 1 | 0,6 | 1 | 0,2 | 1 | 0,2 | 0,2 |
| Version & Change control | Is there a controlled process to avoid accidentally unwanted changes for happening? | 10% | 5 | 1 | 5 | 1 | 5 | 1 | 1 | 0,5 | 0,1 | 0,5 | 0,1 | 0,5 | 0,1 | 0,1 |
| Version & Change control | Is it possible to comment and discuss on specific changes? | 10% | 5 | 3 | 5 | 1 | 5 | 1 | 1 | 0,5 | 0,3 | 0,5 | 0,1 | 0,5 | 0,1 | 0,1 |
| Tool integration | Is it possible to integrate other tools? Is there many tools supported? | 75% | 4 | 3 | 5 | 1 | 4 | 3 | 1 | 3 | 2,3 | 3,8 | 0,8 | 3 | 2,3 | 0,8 |
| Tool integration | Is there a public interface where custom tools can be integrated through? | 15% | 4 | 3 | 3 | 1 | 4 | 4 | 1 | 0,6 | 0,5 | 0,5 | 0,2 | 0,6 | 0,6 | 0,2 |
| Tool integration | Can this tool be integrated into other tools? | 10% | 3 | 3 | 3 | 5 | 3 | 4 | 1 | 0,3 | 0,3 | 0,3 | 0,5 | 0,3 | 0,4 | 0,1 |
| Import | Is it possible to import existing documentation/requirements? | 50% | 3 | 2 | 3 | 1 | 4 | 1 | 1 | 1,5 | 1 | 1,5 | 0,5 | 2 | 0,5 | 0,5 |
| Import | Is importer good at recognizing text, structures, formatting, etc. and convert correctly? | 50% | 4 | 2 | 4 | 1 | 4 | 1 | 1 | 2 | 1 | 2 | 0,5 | 2 | 0,5 | 0,5 |
| Formatting, Multimedia & External files | Is it possible to include external files (multimedia, pictures, documents, ...)? | 60% | 4 | 3 | 4 | 3 | 4 | 2 | 3 | 2,4 | 1,8 | 2,4 | 1,8 | 2,4 | 1,2 | 1,8 |
| Formatting, Multimedia & Document Proofing & | Is there a wide variety of formatting options? | 40% | 3 | 4 | 4 | 3 | 4 | 3 | 3 | 1,2 | 1,6 | 1,6 | 1,2 | 1,6 | 1,2 | 1,2 |
| Document Proofing & Generation | Is it supporting spell checking? | 25% | 4 | 4 | 4 | 1 | 3 | 3 | 1 | 1 | 1 | 1 | 0,3 | 0,8 | 0,8 | 0,3 |
| Document Proofing & Generation | Can it generate documents for use both internal and external (customers, suppliers)? | 30% | 4 | 3 | 4 | 1 | 4 | 1 | 1 | 1,2 | 0,9 | 1,2 | 0,3 | 1,2 | 0,3 | 0,3 |
| Document Proofing & Generation | Is it possible to generate documents based on sub-set of project? | 20% | 5 | 2 | 5 | 1 | 5 | 1 | 1 | 1 | 0,4 | 1 | 0,2 | 1 | 0,2 | 0,2 |
| Document Proofing & Generation | Is it possible to have different visual views of the project (requirement view, document view, model view, ...)? | 25% | 5 | 1 | 4 | 1 | 5 | 1 | 1 | 1,3 | 0,3 | 1 | 0,3 | 1,3 | 0,3 | 0,3 |
| Collaboration | Is it possible for multiple user to collaborate? | 50% | 4 | 4 | 5 | 2 | 5 | 3 | 2 | 2 | 2 | 2,5 | 1 | 2,5 | 1,5 | 1 |
| Collaboration | Is it possible to lock sections to prevent other users from changing? | 30% | 2 | 3 | 2 | 1 | 5 | 1 | 1 | 0,6 | 0,9 | 0,6 | 0,3 | 1,5 | 0,3 | 0,3 |
| Collaboration | Is live tracking/collaboration possible? To see what others is doing? | 20% | 1 | 4 | 2 | 1 | 4 | 1 | 1 | 0,2 | 0,8 | 0,4 | 0,2 | 0,8 | 0,2 | 0,2 |
| User interface & Usability | Is the tool in general user friendly? | 20% | 3 | 4 | 4 | 2 | 3 | 3 | 3 | 0,6 | 0,8 | 0,8 | 0,4 | 0,6 | 0,6 | 0,6 |
| User interface & Usability | Is it possible to show information in different views (requirement view, document view, model view)? | 50% | 5 | 1 | 4 | 1 | 5 | 2 | 3 | 2,5 | 0,5 | 2 | 0,5 | 2,5 | 1 | 1,5 |
| User interface & Usability | Is the user interface intuitive and self explanatory? | 30% | 3 | 4 | 4 | 3 | 3 | 4 | 3 | 0,9 | 1,2 | 1,2 | 0,9 | 0,9 | 1,2 | 0,9 |