



OULUN YLIOPISTO  
UNIVERSITY of OULU

# Evaluation of Stimulus tool in Nokia context

University of Oulu  
Faculty of Information Technology and  
Electrical Engineering / UNIT  
Master's Thesis  
Ville Alakiuttu  
3/2022

## Abstract

The purpose of this thesis was to evaluate Stimulus requirements evaluation tool. This evaluation was done at Nokia to see if the tool is suitable for Nokia's use cases. The thesis used design science research as research method. As the research artifact requirements set was used. This requirement set was then demonstrated using Stimulus tool. The data was collected from workshops held between team from Dassault and team from Nokia. The participants in these workshops were software engineers and requirements engineers. The result of the study was that Stimulus had a good number of useful features but also issues were found. Further evaluation of the tool is needed to determine whether all the issues can be fixed to make the tool suitable for Nokia.

### *Keywords*

Requirements engineering, Stimulus, formal requirements, requirement validation, requirement simulation

### *Supervisor*

University lecturer, PhD Pertti Seppänen

## Foreword

This thesis was written in co-operation with Nokia. I would like to thank Nokia and my colleagues at Nokia for the support and opportunity to write this thesis. I would also like to thank my supervisor from the university side Pertti Seppänen for your expertise and patience with this thesis work.

## ABBREVIATIONS

PTP Time Precision Protocol

UTAUT2 Unified Theory of Acceptance and Use of Technology second version

UML Unified Modelling Language

DOORS Dynamic Object-Oriented Requirements

MBSE Model-based system engineering

# Contents

Abstract .....	2
Foreword .....	3
ABBREVIATIONS.....	4
Contents.....	5
1. Introduction .....	6
2. Prior research.....	8
2.1 Software requirement engineering .....	8
2.2 Requirement engineering formalization and standards.....	9
2.3 Requirement quality.....	10
2.4 Requirement evaluation methods.....	11
2.5 Tools	13
3. Research methods .....	16
3.1 History.....	16
3.2 Implementation .....	18
4. Research Design .....	23
4.1 Literature Study.....	23
4.2 Empirical research.....	25
5. Research and findings.....	27
5.1 Problem identification and motivation.....	27
5.2 Verification artifact of the solution.....	27
5.3 Design and development of the artifact .....	28
5.4 Demonstration.....	31
5.5 Results .....	34
5.6 Evaluation .....	41
6. Conclusions .....	43
References .....	44
Appendix A. Requirements .....	49

# 1. Introduction

Requirement's engineering is one the most important parts of the software development process. As information systems grow larger and larger, getting more complex than ever and, involving multiple stakeholder groups the importance of requirements engineering becomes more important. (Kelanti, 2016)

Need for formalization of requirements has been identified and talked about for decades. Formalization of requirements means use of formalized language in the requirements presentation like Vienna Development Method or Z, or the use of semi-formal languages like UML in the requirements presentation. These languages give rules for the language used in the requirements. Natural language that is most often used for requirements does not limit or give rules to what kind of form or wording should be used in the requirements. (Osman & Zaharin, 2018) This causes multiple problems in the requirements. One of the biggest problems being ambiguity. The requirements need to be based on relevant information gathered from stakeholders and other domain knowledge and written in a way that leaves no room for different interpretation than what the requirements engineer has meant. (Ali, Ahmed, & Shafi, 2018)

There are multiple different standards for different aspects of requirements engineering. there are multiple standards for the requirements specification process and documentation. Standards for evaluating requirements management and evaluation tools. But there is a lack of standards for requirements language and notation. (Schneider & Berenbach, 2013)

Evaluation of requirements can be done manually or by tool. There are multiple methods for both manual and tool-assisted requirement evaluation. One manual way is requirement reviews and one tool-assisted way is simulations. In requirement reviews inspectors will go through the requirements in a systematic way and try to find missed areas and conflicts between the requirements. Simulations are a tool-assisted method of evaluating requirements. there are multiple different types of simulations. One example is doing test runs with the requirements with different inputs to see if the outputs match and if there are conflicts in the logic. (Wallace & Fujii, 2002; Dassault Systèmes, 2021)

Stimulus is a requirements evaluation tool. Stimulus uses simulations to find conflicts and gaps in requirements. This is done with program like runs with the requirements. Stimulus uses models, mainly internal block diagrams to describe the system. The requirements are attached to the internal block diagrams blocks so that the blocks seemingly work as the real program would. The requirements have to be very detailed low-level functional requirements. Nokia was interested in this tool as a need for such tool has been identified.

Dassault of course could theoretically say that this is a good tool for Nokia, and they of course had advertisement material that shows that the tool has benefits and is suitable for Nokia's needs. But Nokia wanted to see how the tool performs in real-world scenarios and what are the potential benefits in the real-world for Nokia.

The validation of Stimulus tool was part of a bigger MBSE (model-based system engineering) exercise at Nokia where new workflows using MBSE were developed for

Nokia. The benefits of Stimulus needed to be seen in Nokia context to better understand how it could be utilized and included in the workflow.

The data for this thesis was collected from demonstration and workshops. To demonstrate the capabilities of Stimulus set of requirements and model of the system were made by Nokia. These were then provided to Dassault. Team at Dassault then made the simulation model in Stimulus based on the provided material. In the workshops knowledge about Stimulus and the demonstration system were exchanged, Nokia team assisted with the creation of the simulation model, and results were discussed.

As result of the workshops and demonstration were that in the current state Stimulus has significant problems that need to be resolved before the tool can be implemented at Nokia. These problems included poor integration with other Dassault products, issues in translating the requirements, and high effort estimates.

As a conclusion from this thesis is that further development of Stimulus needs to be done and further study to workflows to support Stimulus needs to be done.

## 2. Prior research

In this chapter prior research regarding software requirement specification, requirement evaluation, and requirement tools will be discussed. About software requirement specification What kinds of standards and guides there are regarding requirement specification, different levels of formalization in requirement specification, and quality of requirements will be shown. Requirement evaluation the main topics will be to show different verification types and methods and focus more especially on automated requirement verification methods. In requirement tools I will focus on requirement verification and evaluation tools rather than requirement management tools.

### 2.1 Software requirement engineering

Aerospace engineering has been one of the main drivers in the development of requirements and requirement specification. Good requirements and requirement specification are crucial in ensuring safety in safety-critical systems. (Wiels, Delmas, Doose, Garoche, Cazin & Durrieu, 2015) Modern information systems are often very extensive and complex systems. Building these systems requires participation of multiple stakeholder groups. There are many steps in software development one of them being requirements engineering which is the heart of software development. (Kelanti, 2016)

Software requirement specification is the core document where requirements are identified, defined, and verified. Software requirement specification is the most important document in the software development life cycle as it forms the base for the software. (Osman & Zaharin, 2018) The importance of requirements specification has been identified by many studies but there are still major problems with requirements specification practices and understanding the importance of requirements specification in organizations. (Fanmuy, Fraga, & Llorens, 2012) Other problems identified for software requirement specification include incorrect and ambiguous requirements, incomplete, inconspicuous, or unmanaged software specification document. The problems in software requirements specification may come expensive and timely for organizations as it has been identified that problems in requirement specification can lead to cost and time overruns. It is crucial that the potential problems in the software requirement specification are found early on in the project as time goes on it becomes more costly to solve and fix these problems. (Osman & Zaharin, 2018) Part of the requirements engineering is requirements specification. Requirement engineering is the process or activity where identification, extraction, analysis, verification, modelling, and the specification of the requirements are done. These requirements are then collected into the requirements specification document and the implementation phase of the software or hardware may begin. (Shah & Jinwala, 2015)

Zave and Jackson (1997) in their article “Four Dark Corners of Requirements Engineering” talked about how requirements need to be refined with the domain knowledge in requirements engineering to make implementable specification. With this they meant that people writing requirements must have information and knowledge about the domain in which the system is being designed to. Implementable in this case means that the specification should not include features that are not executable by the system under design and replacing them with features that the system can fully support.



In the same article Zave and Jackson (1997) talk about that all requirements should contain information about the environment. This ties into the domain knowledge to be used in the specification. If optative statements often called requirements are used to describe the environment of the system it is unnecessary to describe the system itself (Zave & Jackson, 1997). ISO/IEC 29148 states that requirements need to be “implementation-independent” meaning that the requirements should not place any unnecessary constraints on the architectural design. (Ryan, Wheatcraft, Dick, & Zinni, 2014)

As already stated, ambiguity has been identified to be a major thread for requirements engineering and requirement specification. About 30 percent of quality issues in requirements were related or directly caused by ambiguity. Another 22 percent of quality issues were related to whether requirements can be confirmed to be fitting and good. (Ali, Ahmed, & Shafi, 2018) One of the major causes for ambiguity in requirements engineering and requirements specification has been identified to be natural language. (Osman & Zaharin, 2018; Shah & Jinwala, 2015) The biggest issue in requirements engineering is to extract correct and most relevant requirements from the users and other given source material. These requirements need to be specified in a manner that is not ambiguous so that they can then be understood by others and the integrity and correctness of the requirements can be verified. (Shah & Jinwala, 2015) In requirements specification process it has been observed in studies that the most common way of specifying requirements are non-formalized and semi-formalized methods. (Osman & Zaharin, 2018; Shah & Jinwala, 2015) Requirements are also often initially natural language during the requirement extraction process this means that even in some projects where formal languages are used some of the information from the requirements are lost in translation or the natural language is miss interpreted. (Shah & Jinwala, 2015)

## 2.2 Requirement engineering formalization and standards

Formal languages like object constraint language, Z, and Vienna Development Method have well-defined semantics and rules that help reduce ambiguity as they leave less to be interpreted by the reader. (Shah & Jinwala, 2015) Also, semi-formal languages like UML share some of the same advantages as formal languages while being slightly more flexible. Non-formal, natural language meaning English or other similar language made by human leave a lot to be interpreted by the reader and can easily lead to the requirements or specification be miss interpreted. (Bruel et al., 2019; Shah & Jinwala, 2015) Standards like ISO/IEC/IEEE 24765:2010 system and software engineering vocabulary can make the natural language less “natural” and more formal by introducing some restrictions to the language. (Bruel et al., 2019; Schneider & Berenbach, 2013)

There are software and system life cycle standards like ISO 15288 and ISO/IEC/IEEE 12207 that include and require the use of other ISO standards that focus on requirement specification and verification of requirements. Use of standards has been found to help minimize many of the problems faced during software development and requirements specification. (Bruel et al. 2019; Schneider et al., 2013) These standards are commonly used in system engineering. Many of the commonly used standards have been harmonized and combined by ISO, IEC, and IEEE this is why many of the standards have two or three of these standardization organizations listed in front of them. This has been done due to the complexity of the software engineering domain but causes some confusion in readers. (Schneider & Berenbach, 2013) These tie into a large group of other ISO, IEC, and IEEE standards like ISO/IEC/IEEE 24765:2010 for vocabulary used in software engineering, ISO/IEC/IEEE 24766:2010 which is a guide for evaluating requirement engineering tools. Paredis et al. (2013) have made a mapping

study on how the different software engineering ISO, IEC, and IEEE standards tie together and the predecessors for these standards.

Requirements are derived from a user's needs. ISO/IEC 29148 and INCOSE -TP-2010-006-01 characterise good requirements as necessary, implementation free or implementation-independent, unambiguous, consistent, complete, singular, feasible, traceable, verifiable. (Ryan, 2014)

There are plenty of standards for formalizing software requirements specification and this has been formal specification has been widely accepted. Also, requirements verification and requirements tools evaluation have standards like ISO/IEC 25030 and ISO/IEC/IEEE 29148. However, requirements and requirement notations lack standards. There are lots of notations and syntaxes for requirements but very little in terms of standardisation of them. This causes problems as many tools use their own proprietary notation or languages. This means that when companies develop software, they have to take the language required by the tool and use that. If multiple tools are chosen to be used the requirements have to be translated to the other notations. (Bruehl et al. 2019; Schneider et al., 2013) This translation issue was also present in Walia and Carver's (2009) study about software requirement errors.

ISO/IEC TR 24766:2009 is standard made for requirement tool evaluation. It gives guidance on how the evaluation should be done and what kinds of things the evaluator should pay attention to. This standard is to find out the capabilities of the tool in the given context. This standard is aligned with IEEE 1220 application and management of the system engineering process and ISO/IEC/IEEE 15288:2002 System life cycle processes. According to Schneider and Berenbach (2013) some of the shortcomings of this standard is that it does not consider that most requirement tools today do not come with all of the necessary features out of the box and the tools need to be extended or supplemented. The standard does not provide good guidance for this. Another problem identified by Schneider and Berenbach (2013) was that the standard does not consider toolchains. There is no guidance on how to work when there are multiple requirement tools in use and how these should be linked together.

There are many different requirement languages that have different levels of formalization. Most requirements are written in natural language and other examples of notation types are mathematical and graph-based notations. Natural language requirement notations can be formalized by setting rules and guidelines on the notation. An example of this is Stimulus. (Bruehl et al., 2019) For example in a case where light switch is on and when switch gets turned off the light is turned off. Requirements for this in Stimulus language would be "When light\_switch is ON, light shall be ON" and "When light\_switch is OFF, light shall be OFF". Another formalized natural language notation similar to Stimulus language would be User Stories. User Stories uses template of "As a <user> I want to <target> so that <benefit>". This has one notable difference to Stimulus that this requires a user or an actor, so it is more user-oriented than Stimulus and is, therefore, more suitable for user-oriented designs or collecting user stories. (Wautelet, Heng, Kolp, & Mirbel, 2014)

## 2.3 Requirement quality

Requirement's quality is a big talking point in requirements engineering. In this, one must remember that quality requirements are different from requirement quality. Quality requirements are requirements that define some quality target that the system must meet. (Bøegh, 2008) For software requirements there are quality models that can be used to evaluate quality of requirements. One of these models is called NLSRS

Quality Model. In the model, requirements have two Goal properties, these being requirement sentences quality and requirement document quality. The two Goal Properties have some sub-properties for requirement sentence quality the sub-properties are non-ambiguity, completeness, and understandability. The quality of the requirements then directly impacts quality of requirement specification. (Fabbrini, Fusani, Gnesi, & Lami, 2000) In Fabbrini et al. (2000) study natural language is yet again stated to be one of the major problems for ambiguity and understandability of requirements. Fabbrini et al. (2000) levels of formalization slightly vary from the other literature as they have an additional category of structured natural language on top of the three categories natural language, semi-formal, and formal that the other studies use.

Again, just like in the development of requirements and requirement specification aerospace engineering has been the main driver in developing evaluation methods for requirements and requirement specification.

Errors in requirements must be detected early in the development process as it becomes more and more expensive to fix the issue in requirements as the development work progresses. This creates a need for businesses to have a way to verify requirements. (Terry Bahill & Henderson, 2005) There are many methods for requirements verification. The most commonly used methods are reviews and checklists. These are manual verification methods but there are also automated methods. Automated methods commonly utilize simulations where requirements are tested against an external system where requirements are used as input. (Terry Bahill & Henderson, 2005; Lutz, 1996)

## 2.4 Requirement evaluation methods

In reviews, requirements are published and then inspected by requirements engineers, test engineers, and customer representatives. In the inspection, the requirements are confirmed to be accurate and adequately represent the need that the customer has. This method although simple has been identified to have major flaws. The first problem is that requirements engineers have defined and written the requirements then they inspect. Can their judgement be trusted as they can be blind to their own mistakes and for other reasons effectively sop the issues with the requirements. Secondly, the customer representatives often do not have the necessary knowledge and skills to spot the issues with the requirements. (Fischer et al., 1979) Over the years different improvements to the inspection method have been invented one of which is presented by Ficher et al. (1979) in their study about the requirements verification process. The improvement suggested by Fircher et al. (1979) is System Verification Diagrams in this the requirements are presented in a diagram form. This makes the issues more easily identifiable and in a more easily readable form for the customer representatives. Also, quantitative analytics processes can be applied to the diagrams to verify the completeness and consistency of the requirements.

Inspections can be applied to all documents relating to the software development process. This includes requirements specification documents. There are multiple different methods that have been developed over the years for conducting software inspections. There are differences between the inspection methods and studies over the years have argued which one method and what techniques are the best. (Bernárdez, Genero, Durán, & Toro, 2004) Some argue that a checklist is vital for the inspection process in order to align the inspectors and get the inspectors to focus on the important parts of the software requirements (Bernárdez et al. 2004). Some argue that this leads the inspectors to focus on a narrow range of aspects of the requirements and therefore do not find all of the errors (Bernárdez et al. 2004). Some methods suggest that the

inspections are best to be conducted in groups or pairs others say engineers working as individuals are the best. (Bernárdez et al. 2004)

The word error is also not explicit, and standards give different definitions for the word. For example, IEEE standard 610 gives four different definitions for the word error. (Walia & Carver, 2009)

Wiels et al. (2015) found in their research that reviews can be at least partially automated. In their research the verification worked on two levels first structural errors from the specification were identified by creating instances of the specification, additionally static checkers were used to find the remaining errors. Secondly, it was verified that the specification satisfies all the constraints using metamodels. (Wiels et al., 2015) Singh (2018) has used a machine learning approach to automate reviews and detect false positives from reviews. This machine learning requirements review is based on machine learning techniques used in other textual based validation domains mainly movie and product reviews.

Checklists can be considered to be one kind of review. Checklists are formalized method of inspecting requirements where each requirement is checked against a list of items. The checklist contains items that have been identified to be important or easily missable aspects. Checklists can be made specifically for some software domain. Lutz (1996) has developed a checklist for safety-related errors to be used in mission-critical software. More specifically Lutz (1996) has intended his checklist to be used in the aerospace domain. The checklist includes items like are actions for out-of-range values specified, are cases where the desired input is not received handed and so on.

While reviews and checklists are mainly manual work there are automated options available. Automated verification methods are of course heavily tool depended as the software is what goes through the verification process. One method that automated verification method tackle ambiguity in requirements is converting them from natural language to formal languages. They also evaluate language used and have libraries of words that are considered to be ambiguous. The tools count how many times these words are used and use this to evaluate ambiguity in the requirement specification. Evaluating the language used in requirements can also be used to evaluate the completeness and lack of atomicity. Some automated verification methods require the requirements to be presented in a formal language. These methods usually rely on simulations and comparing requirements to each other to find inconsistencies and completeness of the requirement specification. (Osman & Zaharin, 2018)

Requirement simulations are not a new thing as it has been talked about since the 1980s. In requirement simulators, the language used in the requirement can be evaluated. This is depended on the language and formality of the requirement. The requirements simulation can make a program like run on the requirements to see if there are holes in the requirements i.e., incomplete requirements or lack of atomicity. At the same time, the requirements can be tested with different value inputs for requirements that give are dependent on some value, like something requires minimum or maximum value of x. (Wallace & Fujii, 2002; Dassault Systèmes, 2021)

A more modern example of requirement simulation verification is Pace (2004) where conceptual model and simulation model specification is used to verify the requirements specification.

There are also model-based requirement verification methods. One example of these is Aceituna, Do, Walia, and Lee (2011). Their method transforms natural language functional requirements into a state transition diagram in steps. First, the requirements

are transformed into state transition diagram building blocks where one requirement is one block. These blocks can then be arranged into a diagram using transitions between them.

## 2.5 Tools

There are many requirements tools that use different kinds of notations and use different types of evaluation methods. Of course, not all requirement tools have evaluation capabilities. Tools also have varying degrees of automation. (Bruel et al., 2019) Tools can also be divided into two categories. Some tools are made for requirements management, and some are for requirement evaluation or verification. for example, IBM DOORS falls into the category of requirements management and Stimulus is a requirements verification tool.

Regarding ambiguity in requirement specification, there is a large number of different tools developed for detecting and resolving ambiguity. These can save a lot of resources in the requirements specification process as learning and using language used in the particular domain, requirement structures, and patterns in the language take a lot of time from a human to understand processes, expertise, and knowledge (Shah & Jinwala, 2015). For example, tools like WSD, QuaARS, ARM, RESI, SREE, NAI, SR-elicitor and NL2OCL just to name a few. (Shah & Jinwala, 2015) These tools can be categorised into two categories automated and semi-automated tools. Shah and Jinwala (2015) found that detecting ambiguity in requirements works efficiently and accurately when using machine learning tools that utilize ontology approaches and domain knowledge. The majority of tools do not use these approaches instead they rely on natural language processing tools like Stanford parser and Dowser parser are not up to the job yet to work effectively. If these parsers are not working effectively the rest of the process will not work leading to waste of effort. As these tools rely on fixed pre-sets, they do not work well on requirements that by nature change unpredictably thus creating more ambiguity. But according to Shah and Jinwala (2015) no tool will completely and automatically remove ambiguity from the requirements.

There are also tools that do not have verification features or functionalities for detecting ambiguity but use different formal languages to reduce ambiguity. There are problems with this approach as it reduces the readability of the requirements especially among stakeholders that are not trained for that language. Examples of tools that use formalized natural language would be Stimulus and KAOS. Examples of tools using graph-based notation would be Petri Nets and Statecharts. And examples of tools using mathematical notation would be VDM Alloy, or Event-B (Bruel et al., 2019)

Huertas and Juárez-Ramírez (2012) found that the NLARE natural language processing tool for automated requirements evaluation found 96% of the errors in requirements that human inspection found and 84% of atomicity problems. The differences found were at the time still under investigation. Incompleteness evaluation NLARE tool was still lacking as it only managed to score 56%. This was due to NLARE expected more detail in the requirements than was actually given leading to NLARE's lack of common sense. NLARE checked over 200 requirements per minute while it was estimated that the human reviewer can do less than one requirement per minute. The tool checking is much faster than what a human can do which is a big part of why requirement verification tools are so attractive as they save a lot of human resources. (Huertas & Juárez-Ramírez, 2012)

There are multiple tools for generating UML diagrams from natural language requirements. Some of which are CIRCE, LIDA, and UCDA (Joshi & Deshpande,

2012). Joshi and Deshpande (2012) in their study proposed a new methodology for generating UML diagrams from natural language requirements. This new methodology is Requirement analysis and UML diagram extraction. It works by first using a natural language tool to clean the natural language into new sentences that are confirmed by the tool to be semantically correct. These new requirements are then used to identify concepts and their relationships to form UML presentations.

Traceability is an important part of requirements engineering and requirements tools. Requirement traceability is mandated by many standards like U.S. Department of Defence standard 2167A and MIL-STD-498 standard (Ramesh, Powers, Stubbs, & Edwards, 1995; Ramesh, 1998). Traceability shows that the system requirements follow the stakeholder needs and other design elements follow the requirements. Even though traceability has been identified to be an important part of the system design process. (Ramesh, Powers, Stubbs, & Edwards, 1995) Traceability is also seen as a trend in most commonly used and purchased requirement tools as according to Bruel et al. (2019) requirements tools with traceability support are to most commonly used.

IBM DOORS Next is a requirements management tool made by IBM. DOORS stands for Dynamic Object-Oriented Requirements System. DOORS has the ability of generating reports and publish the reports for reviews. DOORS Next has a traceability system that allows linking requirements with tags and to other requirements. The users can also see different test cases and test coverage of the requirements. Another important feature Of DOORS Next is the ability to see the change history of the requirements. This is a good feature for the reviews as the requirements can be easily rolled back to the previous state if in the review it is seen that the changes were not up to par or not necessary. (IBM Corporation, 2021)

Abbasi, Jabeen, Hafeez, Batool, & Fareen (2015) compared RationalRequisitePro, Objectiver, CaseComplete, RMTrak, Optimal Trace, Analyst Pro, DOORS, and GMARC requirement tools. Assessment attributes used by Abbasi et al. (2015) were glossary, templates, traceability, tool integration, document support, graphical representation, checklist, and scalability. It was not explained why these were chosen as the evaluation attributes. Each tool was evaluated whether or not they have that attribute. The elicitation methods used by each tool was also evaluated. According to Abbasi et al. (2015) Objectiver was the best tool for software requirements.

Similarly, to Abbasi et al. (2015) Sharma and Sharma (2014) have made a similar study evaluating software requirement tools. They also had a similar list of tools. Tools used in the study were RationalRequisitePro, Objectiver, CaseComplete, RMTrak, and Optimal Trace. Also, similarly to the Abbasi et al. (2015) study the attributes used to evaluate the tools were glossary, templates, traceability, tool integration, document support, graphical presentation, and checklist. Much like Abbasi et al. (2015) Sharma and Sharma (2014) found that Objectiver was the best tool for software requirements as it had all of the evaluation attributes. All thought confusingly in this study in the table checkmark is labelled as feature is not available meaning that according to the table Objectiver has none of the attributes.

Stimulus is a requirements verification tool made by Argosim. It has recently been bought by Dassault systems. Stimulus uses simulations as its verification method. Simulation run with multiple input values to see if there are caps or overlap in the requirements. As it requires the requirements to be written in a formal manner it does not evaluate the language used in the requirements. All ambiguities are expected to show up in the simulations results. Stimulus does not support traceability as it focuses solely on the requirements and does not consider other steps of the design process. The tool is expected to be used with a requirement management tool that features

traceability. The simulation also only supports functional requirements and does not cover quality or non-functional requirements. Stimulus offers state machines and block diagrams to help users visualize the requirements and provide additional clarity on the expected behaviour of the system. As Stimulus uses its own proprietary formal language also called Stimulus it risks that non-experts may have hard time understanding the language and requirements. (Bruel et al., 2019; Dassault Systèmes, 2021)

### 3. Research methods

In this chapter, the research methods and implementation of the research methods will be gone through. The research method used was design science research. Snowballing has been used to search for reference material.

#### 3.1 History

In design science research aim is to solve and find a solution to a research problem by developing and using an artifact. This is done in an iterative loop of development and verification cycles. There is plenty of literature on design science research in information systems and the IT field in general.

The idea to make design into science or combine design and science has been around at least since the 1920s. In the 1920s there was a strong movement to find scientific search and architect Le Corbusier described house as a machine made for living. But it was not until the 1960s when design science started to form into an actual methodology. (Cross, 2001)

Design science research has its origins in the 1950s and 1960s. The 1960s space race to launch the first human-made satellite showed that the American scientists and engineers lacked creativity. The new research methodology was developed in a series of meetings and conferences in the 1960s and 1970s held in the UK and USA. There were some notable rejections of the new methodology by some of the original pioneers and researchers of design methods. (Cross, 1993)

The first generation of design science method from the 1960s used the application of systematic rational, and scientific methods as its base. The second generation of design science method used a more practice focus approach in its base. In the second generation the scientist were partners with the problem owners and tried to find satisfactory and argumentative instead of trying to optimize and elevate the scientist above other stakeholders of the problem. These two generations continued to develop independently from each other. (Cross, 1993)

Two more later generations of the design science method although not as pronoun could be considered to exist. First in the 1980s mainly developed and used in AI research and second in the 1990s which was a mix of first- and second-generation design science methods. The generation used in the AI development was hoped to automate and or intelligently assist the design process. The third generation was a mix of the two previous generations and focused to understand the communicative nature of the problem being solved with the design and the communicative nature of the design itself. (Cross, 1993)

IT and information systems are designed to serve a human purpose. These systems are made to be used by humans and improve the efficiency and effectiveness of work in organizations. Computer science does not study natural phenomena but artificial instead. In classic natural sciences like physics, mathematics, and chemistry they use laws, models, and theories to explain reality and the research process involves making



claims about reality and proving them right using norms of truth and explanatory power. (March & Smith, 1995)

Natural science is also viewed as a composition of two activities discovery and justification. While the scientific discovery has no ridged definition, the process of which it happens or is not well understood the general thought is that there is some kind of logic behind it. The scientific discovery in mainstream philosophy is thought to be a creative process much like designing and creating artifacts for design science research. The justification part is thought to be in mainstream philosophy disproving claims and discoveries as a single negative instance can disprove the whole theory. (March & Smith, 1995)

In design science research in the IT field the aim is to create something that serves a human purpose more than explaining reality. Design science products can be four types constructs, models, implementations, and methods. In design science research the scientists can design and study artificial phenomena this is the dual nature of IT research. The IT field is highly innovative and design science research is therefore a suitable research method for the industry. The IT field is also fast-changing, and the research needs, and artifacts change. Also, because the IT field is fast changing it creates artifacts at an ever-increasing rate. (March & Smith, 1995)

Design science research engineering, artificial sciences and natural sciences in its base and it aims to solve a practical problem. Design science research at its core has the artifact. This artifact is created by the researcher. This drives innovation and creativity. These artifacts are based on prior knowledge and are verified by the use of design science methodology. (Hevner et. al 2004)

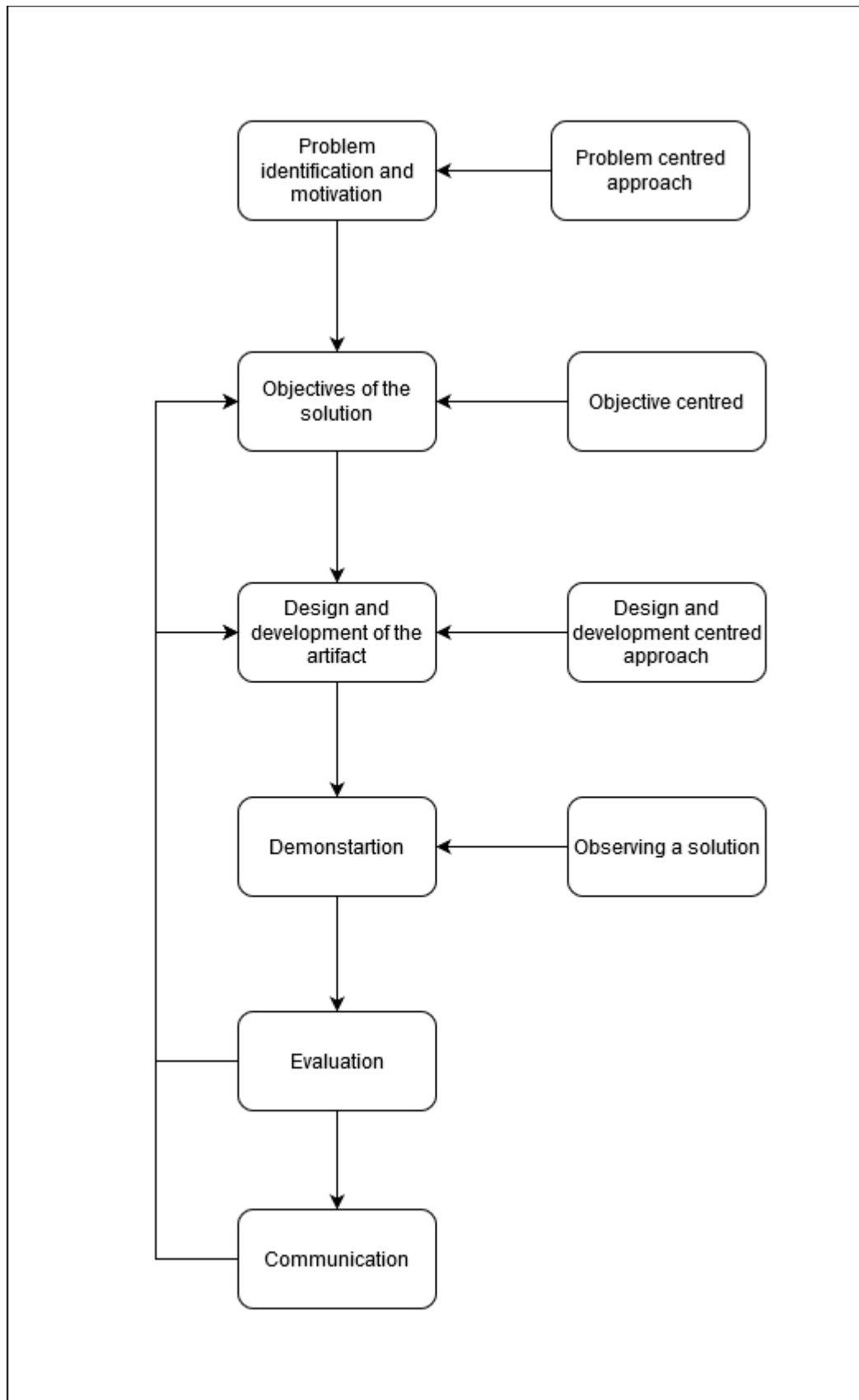
Design science research in information systems is a combination of behavioural science and design science. Behavioural science comes from natural sciences and design science in engineering. Behavioural science is interested in explaining and proving theories that explain or predict human or organizational behaviour. On the other hand, design science research attempts to create artifacts that are innovative and new and in this way push the human and organizational capabilities. So behavioural science tries to explain what is true in human and organizational behaviour and design science research tries to push the boundaries of human and organizational boundaries. (Hevner et al. 2004)

The consensus in the IS research literature is that the research results of an empirical IS study should be something implementable. This leads to the importance of design that is also well recognized in the IS literature. The difficulty in good design is that it is very difficult to design an useful artifact in a field or domain area where there is not a lot of prior research and information on the topic on to base the artifact on. As the domain knowledge is built the artifacts are being built in areas where it was not previously been believed that IT support was possible. (Hevner et al. 2004)

One of the dangers of design science is related to both information system research discussed in the previous paragraph and to behavioural science. If the artifact is well designed and planned but does not have an adequate theory base it dangers of not being applicable to the real world. Hence it is important to line up the artifact with a good theory base and the real world. It must also be remembered that design science research is not just about building an artifact but also rigorous and iterative testing and evaluation of the artifact. (Hevner et al. 2004)

## 3.2 Implementation

In design science research the purpose is to solve the research problem using an artifact. This artifact is a solution to the problem. Design science research process has six steps. These steps being problem identification and motivation, objectives of a solution, design and development, demonstration, evaluation, and communication. (Pfeffers et al., 2006)



**Figure 1.** Pfeffers' design science process

In problem identification the problem is defined, and the importance of the problem and solution is shown. The problem needs to be identified on a level that is atomic enough

to be able to be used to build the artifact to solve. Meaning that if the problem is complex then the problem needs to be understood well to build an effective solution to the problem. Identifying the value of the solution is important as this shows the importance of finding a solution to the problem and motivates the study. (Pfeffers et al., 2006)

In the objectives of the solution the requirements for the solution are defined and the objectives for the solution are defined. How and why the new solution would be better than existing solutions if they exist. How the new solution is supposed to solve the problem and what the objectives are that need to be filled by the artifact. This step may need some extra domain knowledge and knowledge about the existing solutions. (Pfeffers et al., 2006)

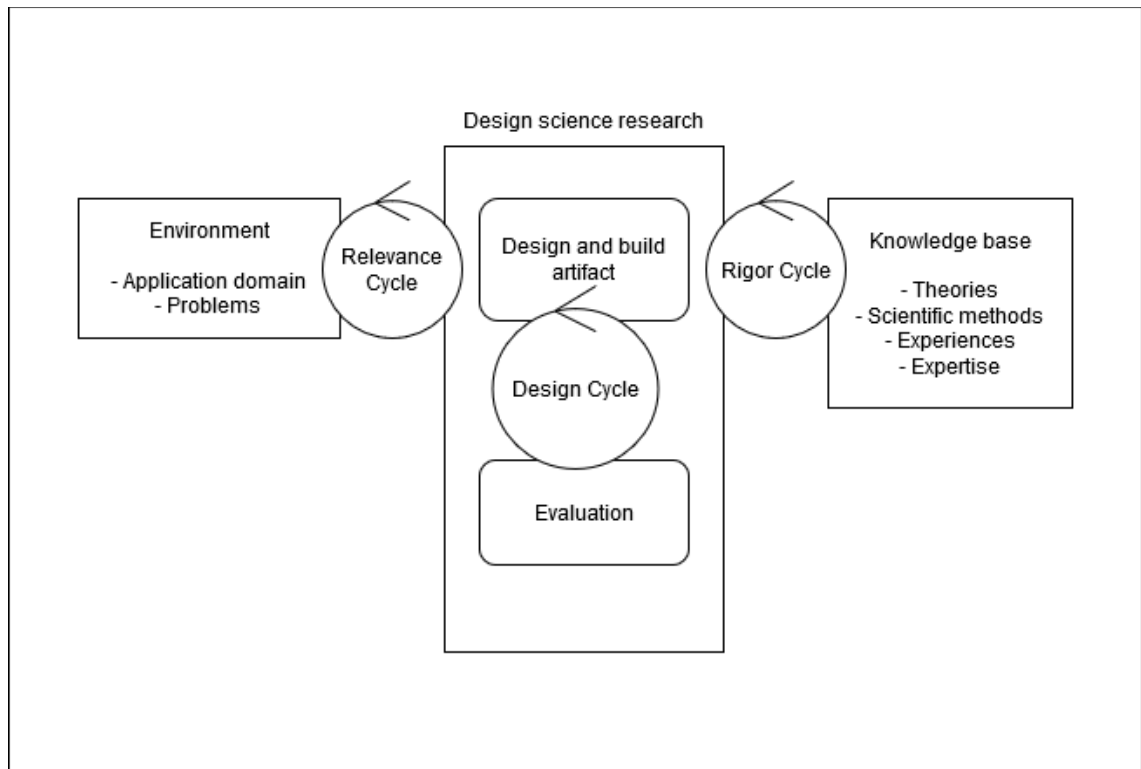
Designed and developed the artifact. The architecture of the artifact and the wanted functionalities of the artifact are made. The artifact is then created. This step requires knowledge about how the defined objectives of the solution turn into design and knowledge about the implementation methods used, coding etc. (Pfeffers et al., 2006)

In the demonstration, it is shown how the artifact is used and how it solves the problem. It is demonstrated that the artifact is an effective solution to the problem. This step can include depending on the solution different methods of demonstration experimentation, simulation, case study etc. (Pfeffers et al., 2006)

In evaluation, the artifact is evaluated. How well and effectively it performs and solves the problem. The findings are then iterated back to the design. The results from the demonstration step are now evaluated. The objectives are needed again in this step as it needs to be evaluated whether the use of the artifact has yielded the desired results that were set as objectives. This may include different evaluation methods depending on the nature of the problem and solution. The effectiveness of the solution can be evaluated in many different ways depending on the solution and artifact these can be for example client feedback, simulations, or performance metrics. After this, the research may be looped back to the design and development step. (Pfeffers et al., 2006)

Communication means reporting the result and the findings. Scientific publication is written and published. It is important to weigh the problem and importance of the problem and solution, the artifact, novelty or utility of the artefact, and the effectiveness of the solution. Standard empirical research process structure (problem definition, literature review, hypothesis, data collection, analysis, results, discussion, and conclusions) may be used for the publication. (Pfeffers et al., 2006)

Design science research process has four possible entry points problem centred approach, objective centred solution, and design and development centred approach. Regardless of the entry point, all steps must be made. In problem centred approach the research loop is started from the first step problem identification. In objective centred solution the research loop is started from the second step objectives of a solution. In a design and development centred approach, the research loop is entered in the third step design and development. In the last possible entry point observing a solution is entered from the fourth step demonstration. Even though the loop may be entered from later than the first step it is possible to loop back to steps two or three from steps five and six. (Pfeffers et al., 2006)



**Figure 2.** Hevner's design science process

Hevner et al. (2004) give seven guidelines for conducting design science research in the information systems domain. The first guideline is that as the product of design science research there should be a usable artifact in a form of construct, model, method, or instantiation. The second guideline is design science research focuses on solving relevant and important business problems and the solution needs to reflect this. The third guideline is that the artifact must be rigorously evaluated. The fourth guideline is that the research must provide clear and verifiable contributions to the field that the artifact operates in. The fifth guideline is research rigor, both construction and evaluation of the design artifact must be done rigorously. The sixth guideline is design as a research process all the domain knowledge and expertise must be used to create an effective artifact. The last and seventh guideline is research communication the research needs to be presented in an effective manner to the readers.

Hevner's (2007) design science method varies slightly from Pfeffer's et al.'s (2006) version. Hevner (2007) divides design science into three parts environment, design science research, and knowledge base. The environment is the domain space that the artifact works in. The environment can consist of people, organizational systems, and technical systems it also contains problems that the artifact solves. Design science research consists of building and designing the artifact and evaluating the artifact. The last part knowledge base is the foundation. It consists of theories and scientific methods. It also contains the expertise and experiences collected from and needed for the research.

Hevner (2007) has three design science cycles design cycle, rigor cycle, and relevance cycle. The design cycle is the cycle between designing and building the artifact and evaluating the artifact. This is the core cycle of any design science research. This strengthens the feedback loop of building the artifact. The artifact is continuously built and tested until it reaches the set objectives.

Rigor cycle is in between the design science research and the knowledge base. The purpose of this cycle is to bring theories, scientific methods, expertise, and experiences to the ongoing research. This ensures that the artifact is innovative. (Hevner, 2004)

Relevance cycle is for showing the improvement that the design brings. In the relevance cycle, the acceptance criteria are made so that the results will be reflected against and the acceptance criteria will show how the improvement can be measured and shown. The results and reflection against the acceptance criteria will then tell if further iterations are needed. In further iterations, the acceptance criterion can fine-tune and improved if needed. (Hevner, 2004)

## 4. Research Design

### 4.1 Literature Study

The literature study of this thesis was carried out using backward snowballing combined with database searches.

Jalili and Wohlin (2012) found that snowballing was very effective in terms of finding relevant papers compared to database searches. In database searches, 85-percent of the result papers were found to not be irrelevant and only 32-percent of the snowballing search results were found to be irrelevant. For this reason, snowballing was chosen as the method for finding references in this thesis. However prior experience of the researcher heavily affects this as knowledge of the domain helps identify keywords and activities from titles and knowing key authors in the area is also beneficial in finding relevant articles. Snowballing can be done both backwards and forwards and Jalili and Wholin (2012) recommend using both backwards and forwards snowballing in the study. Jalili and Wohlin (2012) state that guidelines do not recommend using only forward snowballing in the study, but it should be paired with backward snowballing. In backward snowballing the idea is to go through the reference list of articles to find further references. Forward snowballing means that a database is used to see which articles use this article as a reference. Wohlin (2014) added that context should be taken into count when snowballing and not just the reference list meaning that the articles should be read and seen where articles have been referenced and what was written in the article about the reference. This according to Wohlin (2014) helps identify the relevant articles. One of the drawbacks of snowballing identified by Jalili and Wohlin (2012) was that authors usually use their own papers or papers that they have been part of as references which may lead to research bias.

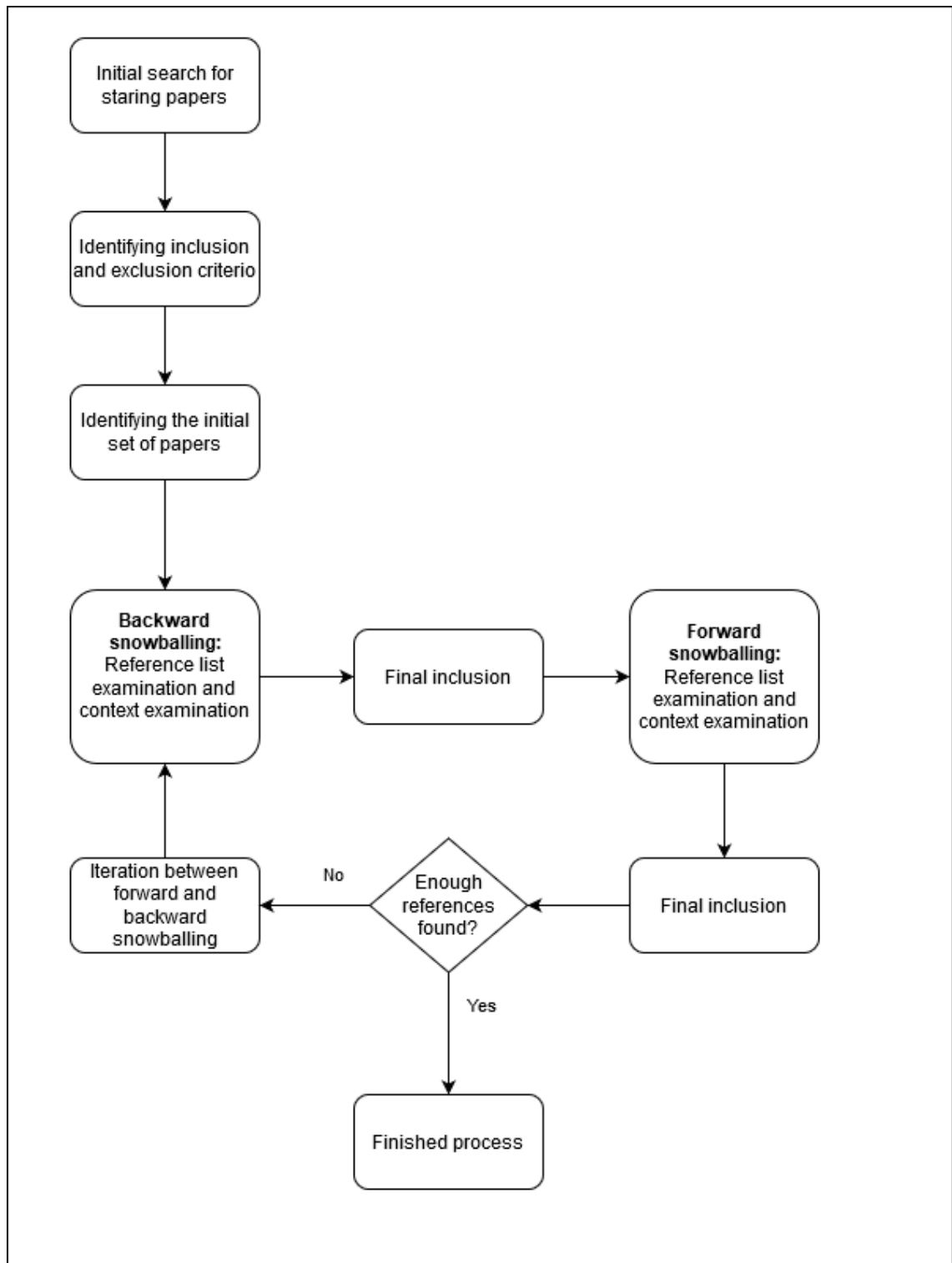
Like for database searches, inclusion and exclusion criteria need to be decided before starting the snowballing. This criterion is used for both forward and backward snowballing.

In database searches, the first step is to formulate a suitable search string and identify relevant keywords. Snowballing starts from an initial set of articles. For search of the starting set, Google Scholar is recommended by Wohlin (2014) to avoid bias towards any publishers. It is important that all of the communities that have made relevant findings and articles on the subject get represented. The size of the starting set should be chosen accordingly to the size of the subject being researched. The starting set must not be too small. The starting set needs to be diverse and include papers from different publishers, authors, and years. Wohlin (2014) also states that there is no golden path to finding the best starting set and similar challenges are faced in this as in formulation search strings for database searches.

After the initial starting set has been established. The snowballing can then be started. First going through the reference lists and excluding papers not matching the basic inclusion criteria and papers that have already been gone through. After this basic exclusion, the rest of the papers are candidates for included references. Next, the papers that were identified as potential references titles, publishers, publication venues, and authors are examined. If there is no reason to exclude the papers Based on these, they

continue to the next phase. Then abstracts of the articles left are read and then further examination can be done to finally include or exclude the paper. It is not recommended to read the whole paper from start to finish but instead glance over the paper to see if there is some useful information there.

Next forward snowballing can be used on the included papers. In this, the papers that use this paper as a reference are examined. This can be done through Google Scholar where the citations can be seen. A similar screening process is done to this list as was done to the backwards snowballing papers.



**Figure 3.** Snowballing process



In this thesis the inclusion criterion was:

- Must have at least 5 citations
- Must be written in English
- Is not duplicate to already picked article
- Abstract must have mention about requirements
- Discussion part must be linked to requirements
- Full text must be available in the university of Oulu library or must be open access

The initial set of papers was established by searching Google Scholar with the following search terms “formal AND requirements AND specification”, “requirements AND tools”, “requirements AND machine AND reading”, “requirement AND standard OR standards”, “software AND requirements AND formalization”, “requirement AND notation”, “software AND requirements AND quality” “software AND requirements AND format”, and “software AND requirements AND guide OR guidelines”. The first 10 search results were first evaluated based on the title it suggests that the article has valuable information. Then abstracts were read from the remaining articles to identify the articles further. The citations did not matter for these as newer articles were favoured over citations to get the starting point for backwards snowballing as new as possible as the back limit was the year 1990. This initial set had 10 articles.

After the initial set was established the backward snowballing was started also at the same time terminology such as “user story templates”, “axiomatic requirement”, “software requirement tool comparison”, and “requirement notation OR syntax” and standards ISO/IEC 29110, ISO/IEC/IEEE 12207, ISO/IEC 24766, and ISO/IEC 25030 were collected to further refine the search terms and especially on the standards further searches were needed to determine what the standards were. The initial articles’ reference lists were read, and potential articles were identified by title. After this, it was read wherein the text and in what context the references were referenced in. After this the publishing years were checked, citations were checked, and abstracts were read. If the reference fit the criterion it was picked as reference material for this thesis. This yielded 27 articles after duplicates were removed. This also includes 3 articles that were in the initial starting set of articles. Further 7 articles were gathered from additional database searches. These searches included “software AND engineering AND specification AND natural language processing OR NLP”, “ISO/IEC 29110”, “ISO/IEC/IEEE 12207”, “ISO/IEC 24766”, “ISO/IEC 25030”. Further 5 articles were recommended by the supervisor.

## 4.2 Empirical research

The research method used in this thesis was design science research. This research method was selected because it was seen as the most fitting research method for this application. The design research method allows demonstration and evaluation of the real-world application of the tool. Nokia wanted to see how the tool performs in real-world scenarios and what are the potential benefits in real-world. It was seen that this way the benefits and weaknesses of the tool can be evaluated the best.

Pfeffers’ et al. (2006) method was selected over Hevner’s et al. (2007) version because it is more straightforward and more fitting to Nokia’s existing workflow. The demonstration part is also important in this thesis as the requirement set is sent to Dassault who will then do the simulations on the tool and show how it was done. Because of this, it is important to focus more on the demonstration to then be able to

properly evaluate time and effort spent on the simulation which is one of the main objectives for the research.

This research started with problem identification. Nokia had already acknowledged some problems with the existing requirements validation process and was interested in testing the new tool. After the problem was identified prior research and state of the art in requirements engineering domain was studied. This already answered some of the questions identified in the problem identification like is there a standardized language for machine-readable requirements. The objectives for the solution were then set. These were based on the needs identified by Nokia and the problems identified previously.

The empirical research part of this thesis was carried out shortly as follows. A requirement set was created that was then be sent to Dassault for feedback and simulation. The requirement set was built in team meetings and smaller individual tasks. The requirement set was then refined using feedback from Dassault. The simulation process and simulation results were demonstrated by Dassault. The simulation results and other data gathered from Dassault were reflected against the objectives and evaluation criterion set for the study.

The results and data gathered from the simulation were then reflected on the objectives set for the solution. Additional evaluation of the design and build process of the simulation was done to determine ease of use and effort estimation.

## 5. Research and findings

In this chapter design, science research process is handled according to Pfeffer's design science process.

### 5.1 Problem identification and motivation

The goal of this thesis was to see if Stimulus tool is a beneficial and suitable tool to be used for requirements analysis in Nokia. The tool could be used to evaluate new requirements. Verifying that they are not overlapping with already existing requirements as currently Nokia has a large number of requirements. This makes it hard to identify overlapping requirements in manual reviews and during the process of creating new requirements. Also, there has been cases in the past where some edge case has not been identified and caused problems later on.

Another problem is that there are plenty of cases where there are multiple different choices that lead to a certain goal. Meaning that there can be multiple different options that are Boolean or numerical and these need to be compared to each other to determine what is the final result. For example, if A is true, B is true, and C is false then perform some specific tasks. Here also the edge cases are a risk as there may be some not so obvious combinations that are rarely seen. Because there are many combinations it is hard to manually make sure that all of them have been covered.

Simulating could also ease the load from reviews as manual reviews require lots of resources.

### 5.2 Verification artifact of the solution

The biggest constraint of the solution is time. The solution should not be too complex so that it does not take too long to design, construct, and demonstrate. But complex enough to demonstrate that it is capable of effectively fit into Nokia's domain. Nokia has many complex systems that tie closely into each other. Finding a suitable part of one of them that did not require too much simulating or building the outside system for the demonstration was difficult.

The solution needs to effectively reflect the real world. There was a need for the solution to be as close to Nokia's real scenario as possible so that the true benefits and weaknesses can be seen. Dassault is already able to provide some general examples of how the tool performs but in this case the interest is to see how it performs for Nokia and whether Nokia has suitable use for it and how it performs in Nokia's domain.

1. Show how much time creation of the simulation takes. This way estimation can be done if the benefits out weight the time and effort spent on creating and running the simulation. Also, the time effort compared to the existing requirements review method. Stimulus is not supposed to replace inspections but possibly reduce the amount needed and simulations could be used paired with inspections.

2. Demonstrate the simulation creation process. As the Nokia team did not get to use the tool itself, there was a need to gather as much information about how the tool is used as possible. This is also part of the effort estimation as there is a need to see how easy the tool is to use and how much time would be needed on training Nokia's requirement engineers to use the tool and how many people are needed to effectively use the tool. For example, if the tool is cumbersome to use then more people is needed to build and run the simulations to get the results in a reasonable time.
3. Show what kind of effort is needed to translate requirements to Stimulus language. The requirements are written in natural language and need to be translated in the Stimulus tool to Stimulus language. As already discussed in section 2 translating requirements can lead to loss of information and incorrectly translated requirements may convey wrong information.
4. Show how the requirements are translated. Stimulus tool has a function that will assist in the translation process and Nokia is interested to see how this works in practice. How much domain knowledge does the person doing the translations need to do the work effectively for this it is good that someone outside the company doing the job. This way it can be seen and evaluated how much support and assistance they needed to complete the job.
5. Demonstrate how overlapping and gaps in requirements can be seen. Nokia team was informed by the Stimulus team that this detection of overlapping and insufficient requirements was something that Stimulus was capable of and even good at. Nokia team was already aware that Nokia has some overlapping requirements as few of them were found already when building the artifact.

These objectives are loosely based on the Unified Theory of Acceptance and Use of Technology second version (UTAUT2) framework (Venkatesh, Thong, & Xu, 2012). According to UTAUT2 people's behavioural intentions to use technology is influenced by four key concepts these being performance expectancy, effort expectancy, social influence, and facilitating conditions. Social influence takes into consideration persons gender, age, and experience so, what the user thinks others believe they should use as technology. Facilitating conditions means how the user perceives the support and resources available to perform tasks. Performance and effort expectancy are self-explanatory. They describe the ease of use and what kinds benefits the use of technology will bring to the user. Performance expectancy has been identified to be the strongest of all of the four key constructs (Venkatesh et al., 2012) From the UTAUT2 framework the main interests were in performance expectancy, and effort expectancy. The social influence has little impact on this as users will use the tools that the company provides them and facilitating conditions Nokia is working on new improved working methods and this is a commercial product so support to use the technology will be of course provided by internal and external support personnel.

### 5.3 Design and development of the artifact

Discussions with Dassault had already begun during the problem identification phase on how evaluation of the tool would be carried out. The end result of these discussions was that Nokia will create requirements set and send it to Dassault where employees will then create the simulation and then demonstrate it to Nokia. So, Nokia did not get any evaluation licenses from Dassault and Nokia team did not get any first-hand use of the tool.

Evaluation of Dassault's Stimulus tool is done using a set of requirements. So, the artefact in the thesis was the set of requirements and it was demonstrated using the Stimulus tool the results from the tool were then analysed.

Creation of the requirement set started from identifying suitable feature that the requirements would be derived from. In this case, feature means some functionality of a software system that satisfies the requirements. The feature that was chosen to be used was point-to-point connect profile that supports ITU-T G.8275.2 standard. Feature name “ToP Light phase & time synchronization support (IEEE 1588 & ITU-T G.8275.2)”. This feature adds a new synchronization profile that enables users to deploy networks according to ITU-T G.8275.2 to networks where the backhaul only meets ITU-T G.8265.1 standard.

The requirements set that was used as the artefact for design science research used in this thesis was created in team discussions and smaller individual tasks. First, the feature was chosen where the requirements would be made from. This feature was already derived from a customer need and a feasibility study had been conducted. The main criterion for picking the feature was that it had to be from a real customer need, had to be already scheduled for implementation somewhere next year, and had to be small enough to be demonstrated in the given timeframe for this study. After the development of the requirements set started by going through the feasibility study and the feature was about support for standard documentation for this standard was read. These gave an understanding of what new would have to be created on top of the existing software and the required domain knowledge. Based on this crude system requirements were identified. These were then refined and chopped into smaller requirements. Some base and surrounding requirements were needed from the existing database to give some context for the Dassault team. One of the team members was tasked to search the existing database for requirements that apply to this feature. This served as a base for the feature. The requirements were on the level of “Network two-way packet delay must not exceed 125us” and “5G FDD (handover) shall be able to maintain phase and time synchronization accuracy within 250us”. The “legacy” requirements were a little more detailed and this delta feature mainly relied on the existing requirements and did not add much on top of them. These legacy requirements were taken from the existing requirements database. They were modified to fit this use case and as this study is part of a bigger model-based system engineering pilot where new workflows and work practises were tested the requirements were also needed in this.

It was already agreed at the beginning that Dassault would assist Nokia team with the creation of the requirements set so that Nokia would have a better understanding of what is required from the requirements so that they can be simulated using the tool. The requirements would be sent to Dassault and the team there would give feedback on how to develop the set further and then when a suitable set for simulation is established build and run the simulation and share the process and results with us. The first set of requirements was sent to Dassault and the result of the meeting held between the Dassault and Nokia team was that these requirements were too high level. The key was that requirements for Stimulus tool need to have clear inputs and outputs. For example, interface requirements where it is clearly stated what kind of data the interface provides as output and what are the clear inputs for the interface. As this feature is a delta feature it provides little in terms of such functionality instead of relying on already existing interfaces and data. The requirements set was then refined based on the first meeting with Dassault team.

The simulation was designed and built by Dassault team based on the requirements given. As Dassault’s knowledge about the Nokia domain was limited discussions were held where the necessary knowledge was shared in order to build the simulation correctly.

Some examples of the previous existing requirements “Support for PTP slave functionality according to ITU-T G.8275.2 unicast standard profile” “Phase synchronization mode must have 120s tuning cycle length with PTP as synchronization reference” “Initial coarse tuning cycle length must be parametrized to following list [12s, 60s]” “Coarse tuning packet selection window max length must be 1 tuning cycle with Normal PTP profile” “Coarse tuning packet selection window max length must be 10 tuning cycles with Custom PTP profile” “Normal tuning packet selection window must be 5 tuning cycles”.

The first meeting with Dassault showed Nokia that the requirements need to be much more detailed. They advised Nokia team to draw a UML state machine of the system to help identify the inputs and outputs of the system. A new set of requirements was created based on the previous that split the system requirements into smaller functional requirements. The target piece of the system where the requirements were created from was also chosen to be such that clear input outputs can be shown in the requirements. Also, for this illustrative picture (Figure 4.) below can be seen for a better understanding of the system. This set of requirements can be seen in appendix A.

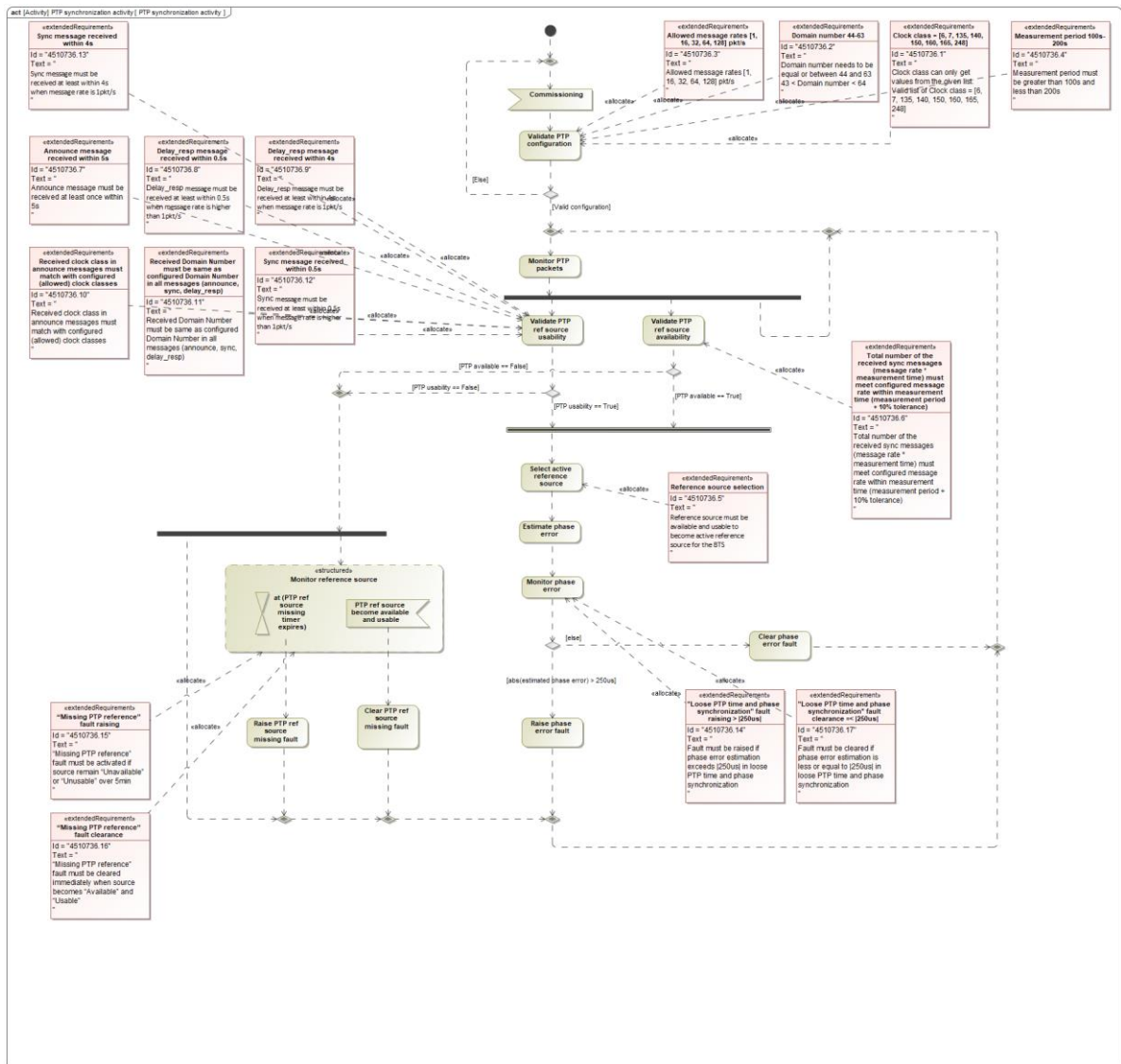


Figure 4. Illustrative picture of the target system

The requirements were sent to Dassault for the simulation process. Dassault arranged a meeting where the requirements were gone through and they gave Nokia team feedback

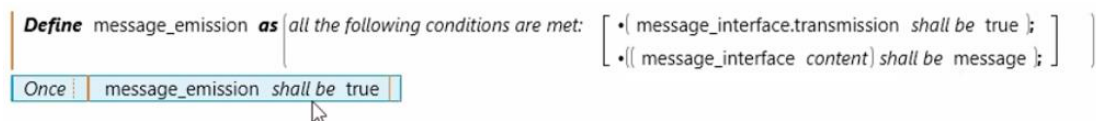
on how the requirements needed to be refined in order to simulate them. The problem with that set of requirements was that it was system-level requirements, but Stimulus is only capable of simulating functional requirements with clear inputs and outputs. The new set with more detailed requirements were then sent again to Dassault for feedback.

The requirements are stored in DOORS Next requirements management tool which is used by Nokia. Dassault also offers their own requirements management system. The requirements are created in DOORS and the language used for the requirements is natural language. This creates challenges in requirements management in Stimulus as Nokia did not know how Stimulus is going to show changes in the requirements. For example, if a set of requirements are translated and then changed in DOORS how can it be seen which requirements the change affects in Stimulus and do the changes have to be done manually or can Stimulus do the changes automatically.

## 5.4 Demonstration

Dassault builds and runs simulations and then demonstrates to Nokia how it worked and how it was done. To help assist Dassault with the creation of the simulation model SysML activity diagram was drawn (Figure 4) of the target system to show where and how the requirements were used in the system.

The requirement set (Appendix A) sent to Dassault was used in the simulation. The requirements were translated to Stimulus language in the Stimulus tool. The requirements will be stored in DOORS Next in a normal situation but in this case to ease the communication between Dassault and Nokia the requirements were shared by email. Stimulus assists with this translation process by providing a library where partly done sentences can be picked (figure 5) that are missing actors and objectives that can be picked from another library that contains all the parts of the system that the user has configured. For example, a sentence like in figure 5 “message” shall be sent through “message interface” is a ready-made sentence structure where the user needs to drop the desired message and message interface that the user has previously configured. In the menu in figure 6 message and message\_interface are examples of interfaces that can be used to fill in the requirement sentence. These interfaces connect the PTP Monitor Block and Fault Manager block. The sentence templates can also be customized or created by users. Template use is not mandatory, and the requirements can be typed out if there is no suitable template for that requirement.

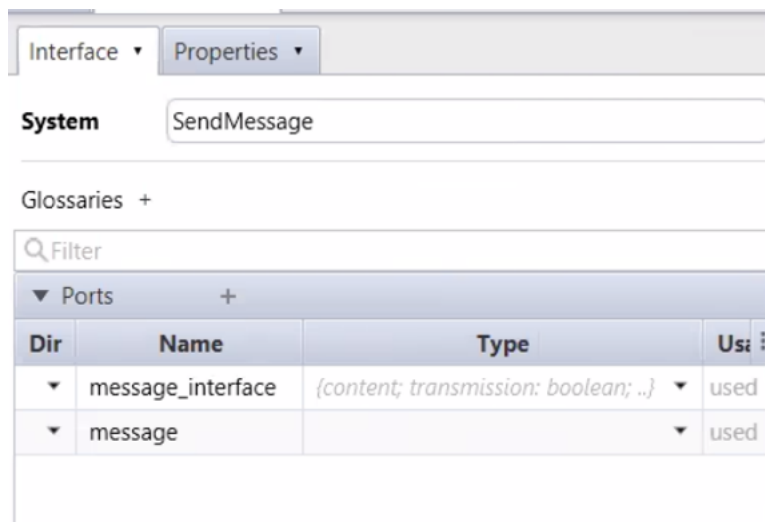


```

Define message_emission as all the following conditions are met: [ •( message_interface.transmission shall be true ; ] [ •([ message_interface content] shall be message ; ] ]
Once message_emission shall be true

```

**Figure 5.** Requirement creation in Stimulus



**Figure 6.** Interface library

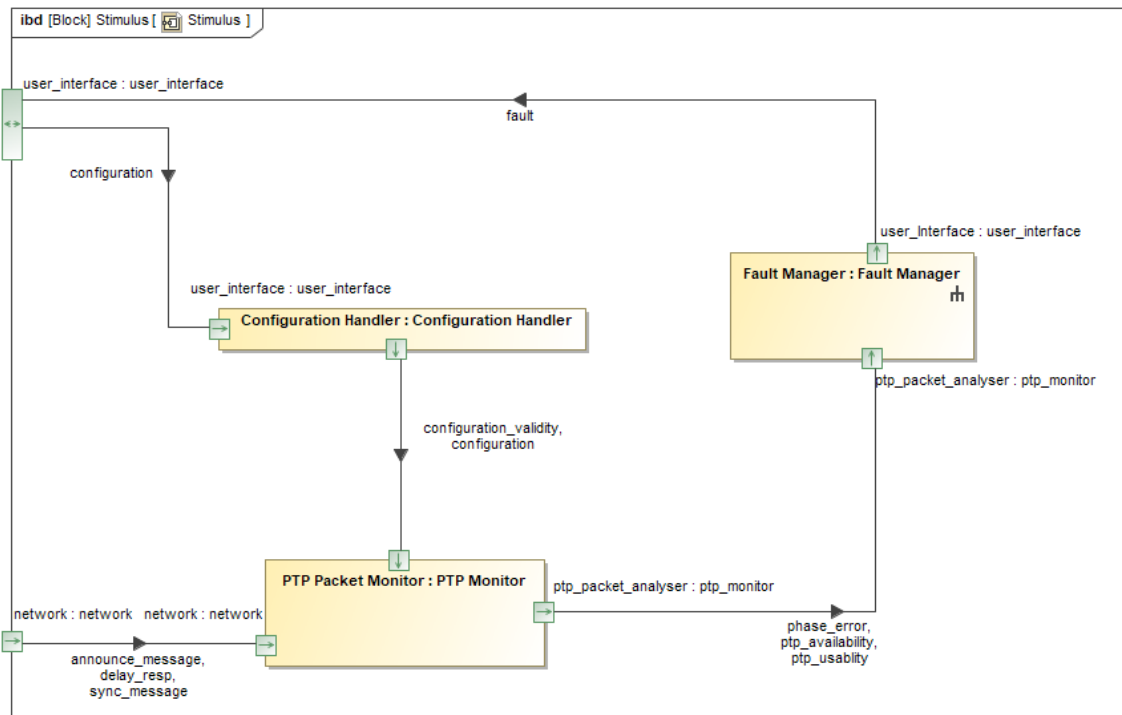
Additionally, to the requirements test environment must be created. This environment is SysML internal block diagram which can be imported to Stimulus or created in the tool itself. In this case, the automatic import did not work and figure 6 had to be manually created in Stimulus. The requirements are then attached or included in the blocks. This way either part of the test environment can be simulated as in a single block or the whole system. This way the interfaces can be tested too. This feature was also demonstrated by Dassault for Nokia. They also showed how the blocks can be reused for creating quick individual test systems for the blocks.

As a base for the demonstration internal block diagram as in figure 7 was used. This figure was provided by Nokia, and it included explanations for the system. Dassault created a library containing the relevant glossary and interfaces for this case. The interfaces made were user interface, network, ptp\_packet\_analyser, and configuration\_handler. These interfaces can be seen in figure 7 connecting the blocks and going out of the system. The blocks in the figure were imported into the demonstration in Stimulus.

Inner internal block diagrams were made inside these top-level ones to explain the functions and related requirements of each block. In the demonstration, these blocks contain the translated requirements and the information required for the simulation.

In this demonstration, the ranges were given by Nokia to Dassault. These were included in the inner levels of the provided internal block diagram (figure 7). For example, the configuration that can be seen going in between user interface and configuration handler has three attributes domain number, packet rate, and clock class. The ranges set for these were 44 to 63 for domain number, list of power to 2 from  $2^0$  to  $2^7$  for packet rate, and list of 6, 7, 135, 140, 150, 160, 165, 248 were given for clock class. These values are taken from the network protocol standard.





**Figure 7.** Top-level internal block diagram of the system

The simulation parameters and ranges are automatically generated for the requirements by Stimulus. These parameters can also be manually adjusted or created from scratch. For integer or float values the steps can be adjusted accordingly. How many runs of the simulation can be chosen. Multiple simulations can be run end-to-end with different parameter values for better coverage.

Observers can be set to observe the simulation output and input values. If the observer rules are violated then the observer indicates this. Multiple observers can be set for the simulation with different rules. These observers are created by choosing a block or interface that the user wants to observe then choosing what attribute or attributes the user wants to observe, and the value the user wants to observe. For example, the observer can be set for fault manager to see when both missing reference source and PTP time and phase error faults are not active. This is one of the observers demonstrated to use.

Simulation draws a graph while the simulation is being run. The graph shows the values that the parameters get during the run and the values that the observers get. The graph can be used to observe the moment when the observers are violated or other desired moment during the simulation to see what was where the values of the parameters at that moment. This gives an idea if there is a missing requirement, or the existing requirements are not complete.

In figure 8 there is an example of how the graphs are displayed. In this graph PTP (Precision Time Protocol) availability is depicted. PTP availability can get four states depending on requirements. Where the line is in the picture at roughly 35 seconds PTP availability state first is WaitToRestore and changes to Available.



**Figure 8.** Example of simulation graphs

The test log file is outputted after the simulation is run. This file contains a table of all the parameters used in the simulation and the values the parameters got during the test. This file can be used for further analysis with different tools or manual analysis. This way the test results can also be stored for future reference or be shared with others.

## 5.5 Results

As a result of the demonstration given to Nokia by Dassault, Nokia got a presentation of the demonstration with a recording of it and a report of the Stimulus mode including requirements and details about the model. More data was collected from workshops held between Nokia and Dassault. This data is in a form of notes that were written during the workshops. In the workshops, the model created in Stimulus, requirements in Stimulus and general use of Stimulus were discussed.

There were four workshops in total held. The participants in the workshops from Nokia included three to four experienced system architects with a minimum of five years of experience in the company and of course me. From Dassault's side, the workshops had two to three sales specialists and one to two industry process specialists. All of the Dassault personnel were experienced users of Stimulus and did not have any prior experience with the system presented by Nokia. Nokia personnel had no prior experiences or knowledge of Stimulus but had extensive knowledge of the system. After each workshop meeting between the Nokia personnel was held where the results and events were discussed, and thoughts were exchanged.

Two additional requirements were added to the set to get Stimulus to find a conflict. Requirements 58, and 59 in Appendix A. These two requirements were relatively simple but added enough complexity to the system to cause a conflict that stimulus found.

As the requirement set was created by Nokia without any experience of the tool this issue with the requirements set was not initially known. Nokia team tried to create a set that would show the capabilities as good as possible but as no one from the team had used Stimulus Nokia team could only guess what kind of requirements would show the capabilities the best.

The glossary in figure 9 shows the attributes used in the Stimulus requirements. In figure 10 is one of the requirements from Stimulus where the use of the glossary can be seen in action. The user first chooses one of the requirement templates and then drag and drops the wanted attributes or items into the template. In this case When and Otherwise templates have been used. As parameters configured domain number, configured message rate, and configured clock class have been dropped into the requirement template. This Stimulus requirement says that all the given attributes must be in the given ranges for the configuration received from the user to be valid. Comparing this to the original requirements written in natural language this requirement comes from requirements 1, 2, 48. These requirements define the given attributes. Here is also the first missing requirement as our provided list did not explicitly state that the configuration must contain these attributes. This was not the only requirement that was

added.

Glossary Table							
Name	Type	Dimension	Unit	Default Value	Default Behavior	Enum Encoding	Comment
User_Interface	Structure						
Configured_Domain_Number	integer						Range 0-255
Configured_Message_rate	integer						Range 2'0 - 2'7
Configured_Clock_Class	integer						Range 0-255
Configuration_Validation1	boolean						
Network	Structure						
Announce_Messages	Structure						
Clock_Class	integer						Range 0-255
Domain_Number	integer						Range 0-255
Message_Rate	integer						Range 0-255
Announce_Message_Lost	boolean						
Delay_Resp	Structure						
Received_Domain_Number	integer						Range 0-255
Received_Packet_Rate	integer						Range 2'0 - 2'7
Delay_Resp	boolean						
Sync_Message	Structure						
Domain_Number	integer						Range 0-255
Packet_Integer	integer						Range 2'0 - 2'7
Sync_Message	boolean						
PTP_Availability	Structure						
Phase_Error	Structure						
Phase_error_Value	real						
Phase_error_status	boolean						
PTP_Usability	boolean						
PTP_Availability	boolean						
Fault_PhaseError	boolean						
Fault_Available_and_Usable	boolean						

**Figure 9.** Glossary of the attributes from Stimulus

```

Configurator_Validator_Req
  ( all the following conditions are met:
    [ ( User_Interface.Configured_Domain_Number ∈ [ 44, 63 ] );
      ( User_Interface.Configured_Message_rate is in set [ 1 ; 16 ; 32 ; 64 ; 128 ] );
      ( User_Interface.Configured_Clock_Class is in set [ 6 ; 7 ; 135 ; 140 ; 150 ; 160 ; 165 ; 248 ] ) ]
    When )
    | Configuration_Validation1 shall be true
    otherwise
    | Configuration_Validation1 shall be false
  
```

**Figure 10.** Stimulus requirements for configuration validity

The requirement in figure 10 is relatively simple and easy to understand but the requirements can get cluttered, long, and hard to understand for an untrained eye. As an example, requirement in figure 11. There are lots of people who need to read these requirements some with less knowledge about the system. For example, requirements need to be readable by implementation teams and managers. This is a common problem with formal requirement languages as discussed in the prior research chapter. To improve readability for stakeholders who have not been trained to read Stimulus language the requirements need to be translated to natural language. Translating requirements can cause ambiguity in the translated requirements and translation mistakes where the translated requirement does not fully match the original requirement.

Another issue related to having requirements written in two different formats. This doubles the work while creating and maintaining the requirements as they need to be changed in two different places. Meaning that the same requirement would be in DOORS twice with a different presentation. At the moment Stimulus does not have any function that would assist with the translation work. Also, the suggestion was that only the requirement written in Stimulus would be fixed using the tool and that would replace the requirement written in natural language.

```

Variable Announce_Message_Flag_Lost : boolean Default: false Default Behavior: Stable
Variable Delay_Resp_Flag : boolean Default: false Default Behavior: Stable
Variable Sync_Message : boolean Default: false Default Behavior: Stable
PTP_Monitor_Req
  When ( Configuration_Validation1 is true ),
    Phase_Usability_REQ
      ( all the following conditions are met:
        [ ( User_Interface.Configured_Domain_Number = Network.Announce_Messages.Domain_Number ) ;
          ( User_Interface.Configured_Domain_Number = Network.Delay_Resp.Received_Domain_Number ) ;
          ( User_Interface.Configured_Domain_Number = Network.Sync_Message.Domain_Number ) ;
          ( User_Interface.Configured_Clock_Class = Network.Announce_Messages.Clock_Class ) ;
          ( Announce_Message_Flag_Lost is false ) ;
          ( Delay_Resp_Flag shall be false ) ;
          ( Sync_Message shall be false ) ]
        When )
        | State.PTP_Usability shall be true
      otherwise
        | State.PTP_Usability shall be false
    Delays
      When ( Network.Announce_Messages.Announce_Message_Lost is false ),
        | After 5[s], | Announce_Message_Flag_Lost shall be true
      When ( Network.Delay_Resp.Delay_Resp is false ),
        | After 0.5[s], | When ( Network.Delay_Resp.Received_Packet_Rate > 1 ), | Delay_Resp_Flag shall be true
      When ( Network.Delay_Resp.Delay_Resp is false ),
        | After 4[s], | When ( Network.Delay_Resp.Received_Packet_Rate = 1 ), | Delay_Resp_Flag shall be true
      When ( Network.Sync_Message.Sync_Message is false ),
        | After 0.5[s], | When ( Network.Delay_Resp.Received_Packet_Rate > 1 ), | Sync_Message shall be true
      When ( Network.Delay_Resp.Delay_Resp is false ),
        | After 4[s], | When ( Network.Delay_Resp.Received_Packet_Rate = 1 ), | Sync_Message shall be true
PTP_Availabe_REQ
  Stability
  | State.PTP_Availability is 'Unavailable by default
  Req for MessageRate is met -> WaitToRestore
  When ( ( Network.Announce_Messages.Message_Rate * 110[s] ) is in between [ 0[s], 110[s] ] ),
    Do
    | For 15[s], | State.PTP_Availability shall be 'WaitToRestore
    afterwards
    | State.PTP_Availability shall be 'Available
  otherwise
  | State.PTP_Availability shall be 'Unavailable
  From Available MessageRate is not met -> Losing
  From the time ( ( Last State.PTP_Availability ) = 'Available ),
  When ( ( Network.Announce_Messages.Message_Rate * 100[s] ) is in between [ 90[s], 110[s] ] ),
    | State.PTP_Availability shall be 'Available
  otherwise
  Do
  | For 5[s],
  | State.PTP_Availability shall be 'Losing
  afterwards
  | State.PTP_Availability shall be 'Unavailable
PTP_Phase_Error
  When ( ( State.PTP_Usability is true ) and ( State.PTP_Availability is 'Available ) ),
  | State.Phase_Error.Phase_error_status is true
  otherwise
  | State.Phase_Error.Phase_error_status is false
  | State.Phase_Error.Phase_error_Value shall be 0[s]
otherwise
  ( ( State.PTP_Usability shall be false )
  ( and ( State.PTP_Availability shall be 'Unavailable ) )
  ( and ( State.Phase_Error.Phase_error_status is false ) )
  ( and ( State.Phase_Error.Phase_error_Value shall be 0[s] ) )

```

Figure 11. Stimulus requirements for fault management

```

Req for MessageRate is met -> WaitToRestore
When ((Network.Announce_Messages.Message_Rate * 110[s]) ∈ [0[s], 110[s]]),
  When (((Last State.PTP_Availability) is 'Losing) or ((Last State.PTP_Availability) is 'Available)),
    State.PTP_Availability shall be 'Available
  otherwise
    Do
      For 15[s], State.PTP_Availability shall be 'WaitToRestore
    afterwards
      State.PTP_Availability shall be 'Available
  otherwise
    From the time ((Last State.PTP_Availability) = 'Available),
      Do
        For 5[s],
          State.PTP_Availability shall be 'Losing
      afterwards
        State.PTP_Availability shall be 'Unavailable
    Before the time ((Last State.PTP_Availability) = 'Available),
      State.PTP_Availability shall be 'Unavailable

```

**Figure 12.** A proposed fix to the requirement in figure 11

Figure 11 contains the conflicting requirement if this is run in Stimulus it would create an error that is displayed (figure 13) in Stimulus. The error occurs because there is a situation where PTP availability status is both unavailable and losing at the same time or there is no differencing which one of these states should become active after this state change. As the PTP availability can only get one state at a time this creates a problem as one of them would have to be picked over the other one. Figure 12 is a proposed fix by the Dassault person who made the model in Stimulus and translated the requirements to Stimulus. The last two lines are the lines that needed to be added. On these lines, it is stated that if the previous state was available the next state should be unavailable. This is incorrect as the state after available should always be losing.

There are also two other mistakes that come from translating the requirements. These mistakes can be seen in both figures 11 and 12. Both are on the line ‘When ( ( Network.Announce\_Messages.Message\_Rate \* 110[s] ) ∈ [ 0[s] , 110[s]] ) ,’. The mistakes here being that the original requirement states that sync messages should be calculated and not announce messages. Secondly, this line means that announce message rate times 110 seconds should be in between 0 and 110 seconds. This should be sync message rate times measurement period which is the 100 seconds should be the same or less than the number of sync messages received in 110 second measurement time. This message is likely caused by the translator’s lack of domain knowledge. Even though for the Nokia team the requirement was clear, and some time was spent to make sure that this requirement is written as understandably as possible.

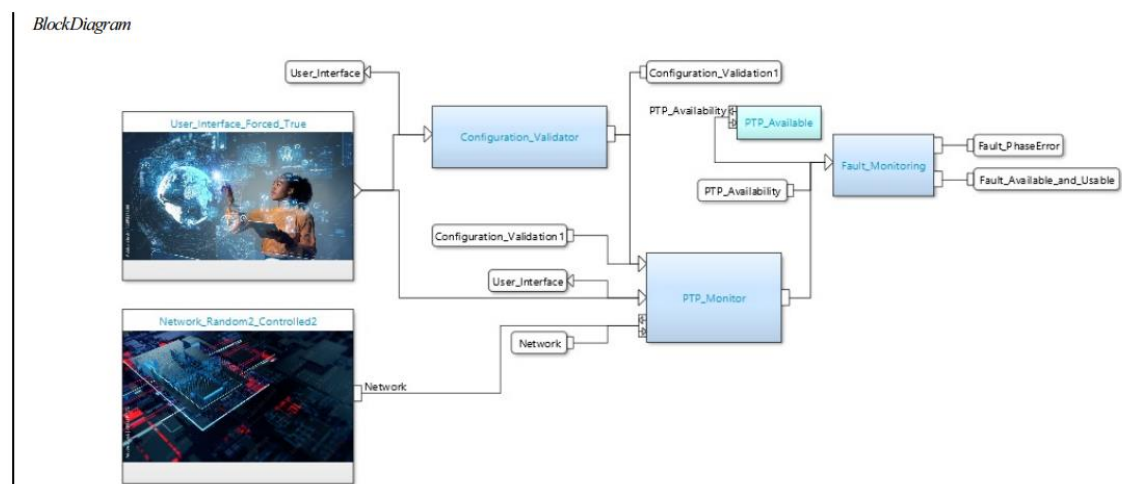
Errors found in requirement in Stimulus are displayed like in figure 13. This is a similar way to which many code compilers and integrated developer environments would display error messages. The error message contains path where the conflict occurs, block where the error occurs, error level if its error or warning, for example, error message explaining the error in more detail, and the step on which the error occurred. In this case, the error occurs on step 5001 of the first run.



System Simulation Errors					
	Path	Name	Error Level	Message	Run/Step
	Requirement:PTP_Monitor	PTP_Monitor	Error	Conflict in PTP_Monitor inside... on variable State.PTP_Availability no value for variable State.PTP_Availability can satisfy the following constraints: State.PTP_Availability = 'Unavailable from State.PTP_Availability shall be 'Unavail... State.PTP_Availability = 'Losing from State.PTP_Availability shall be 'Losing...	0 / 5001

**Figure 13.** Error message in Stimulus

When comparing the internal block diagram (figure 5) that was sent to Dassault to the block diagram in figure 14 they are similar. The only major difference is that the outgoing network and user interface interfaces do not seem to go out of the diagram but are displayed as blocks inside the diagram. These blocks could not be in this case be directly imported from Magic Draw that was used to draw the original picture that was provided to Stimulus team. This was because this feature of Stimulus was still under construction. The requirement in figure 11 would be in the Configuration\_Validator. Based on the input received from user\_interface the Configuration\_Validator block will send Configuration\_Validation1 true or false to PTP\_Monitor. Rest of the blocks work in a similar fashion where they have some input, the input is validated against requirements given for that block, and an appropriate response is sent to the next block based on requirements and input.

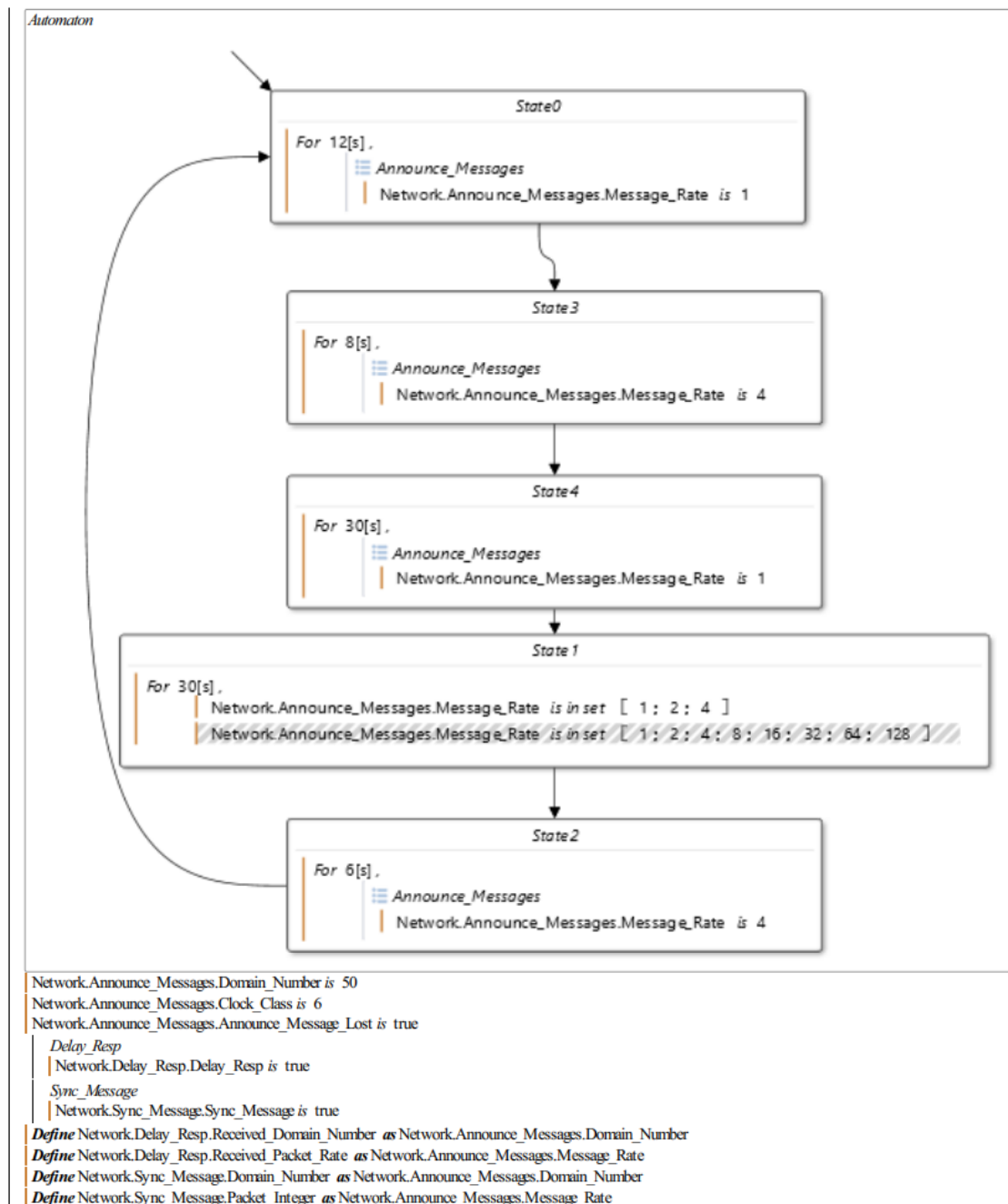


**Figure 14.** Block diagram in Stimulus

Nokia has extensive UML models of the base transceiver station synchronization system that at least at the moment cannot be imported to Stimulus. This creates lots of extra effort as the models would have to be manually drawn in Stimulus. On top of this, there are now two models that have to be maintained.

In figure 15 is a test case made for testing requirement in figures 11 and 12. This test also yields an error in figure 13 when run on requirement 13. Down at the bottom, the default values for the attributes can be seen. If in this stage, the value for the attribute is not given then this value is used. In stage0 where the execution starts the announce message rate is set to 1pkts/s for 12 seconds. In the next stage stage3, the announce message rate is changed to 8pkts/s for 8 seconds. Next stage stage4 message rate is 1 like in the first step. In stage1 the message rate is varied between 1, 2, 4. The greyed text in this stage is 'commented out' so it is not run. The value is randomly changed

every 0.01 seconds which can also be adjusted. In the last step, the message rate is set to 4.



**Figure 15.** Test automation

Here the translation error in the requirements is apparent again. Not only the wrong message is being tested but the message rate here should not be the only variable. The other variable should be the number of received messages. This is because the original requirement was to make sure that the system receives enough messages in a 100-second time frame with the given message rate. The basic random function is not good enough to test the change in the received message rate as it would mean that the message rate changes 100 times in one second. This is unrealistic and would require the



engineer to build simulation sequency to test this variability. This increases efforts as there are lots of these kinds of variations in the system.

Based on the demonstration fixing the issues with the test case is a fast and relatively easy process. Although it is apparent that careful attention needs to be paid to the attributes and logic of the test case to avoid situations like this where a test case is created that is not correct. These kinds of issues are caused by a lack of technical domain knowledge of this specific case.

At first, the major drawback of this demonstration was that the requirement set was not able to show all of the capabilities of Stimulus. No conflicting or overlapping requirements were found. This was due to the nature of the requirement set. As the requirement set was heavily based on a standard that dictated most of the contents of the requirements there was not a change that Stimulus would find some conflict. Synchronization is an area where standards dictate most of the requirements in this regard the requirement set was realistic and true to what the real requirements are, but it was not a good set of requirements to show the capabilities of Stimulus. The requirement set would have had to contain two sets of requirements that modify the same output. The requirements set was also only 57 requirements, so it was still easy to manually check that there were no duplicates.

## 5.6 Evaluation

Reflection against the solution objectives set in chapter 5.2.

The first objective was to show how much time creation of the simulation takes. It took around three weeks from the Dassault person to create the simulation. The exact amount of time spent is not known and it is hard to estimate how many hours in the three weeks were actually spent on the simulation. During the demonstration, the tool seemed simple and reasonably fast tempo to use but on the other hand, the required simulation models in real use would be massive. At the moment as the integration between Magic Draw and Stimulus is not working well this increases the time required to create the simulation as the models cannot be imported from Magic Draw.

The second objective was to demonstrate the simulation creation process. The creation process was demonstrated, and the tool was seen as easy to use but some trainings would be necessary in order to implement the tool. Stimulus had many useful features aiding in the simulation creation process. The missing integration between Magic Draw and Stimulus somewhat hinders this process as the simulation model needs to be drawn by hand even though Magic Draw already contains this kind of diagram. Meaning that the Magic Draw model cannot be directly imported to Stimulus.

The third objective was to show what kind of effort is needed to translate requirements to Stimulus language. The translation process was somewhat troublesome as there were translation errors made during the demonstration. At the moment the effort required is huge as Stimulus does not provide good integration with Magic Draw. As result from this the models have to be drawn and maintained twice in both Magic Draw and Stimulus. The initial effort to translate the huge amount of requirements Nokia has would be a huge undertaking but this was to be expected as Stimulus does not have any automated system for translating bulks of requirements.

The fourth objective was to show how the requirements are translated. Stimulus tool has a function that will assist in the translation process and Nokia is interested to see how this works in practice. During the demonstration, Dassault personnel demonstrated how

new requirements are created using the template system that Stimulus provides to assist with the translation work. The demonstration showed that the person doing the translation work needs to have good domain knowledge of the system. The issue was in translation errors as the Dassault personnel lacked the knowledge of the system, they were not able to successfully translate all of the requirements. The translation work requires precision as errors in the translation also affects preparing the simulation. The proposal from Dassault was that the requirements would be first defined in Stimulus stored in DOORS where the requirements management is done in Stimulus syntax.

The final objective was to demonstrate how overlapping and gaps in requirements can be seen. This was seen in the last two requirements where Stimulus gave an error. Stimulus pointed out in the error where the error occurred and what happened during the run. This was relatively easy to understand for people with programming background as the error message was displayed in a similar manner to compiler errors. Creation of the test cases requires domain knowledge of the system to design. This was seen in figure 15. where the test case was not realistic.

## 6. Conclusions

In this thesis, it was evaluated whether Stimulus requirements evaluation tool is suitable for Nokia. Design science research was used as the research method and as the research artifact a set of requirements were used (Appendix A). This set of requirements was provided to team at Dassault who then demonstrated how this requirement set works in Stimulus. workshops were used to discuss and develop the demonstration.

As result from the study was that Stimulus has a lot of potential and offers a good set of features to evaluate requirements and fix gaps and overlapping requirements. There were problems too with the tool some of which were related to integration with other tools. The requirements and requirement set must be a certain type to be used in Stimulus. For example, only detailed functional requirements can be used and additionally the requirement set has to contain multiple requirements modifying the same output for Stimulus to find problems with the requirements. Stimulus requires its own nonstandard language to be used meaning that either all of the requirements used in the company have to be in Stimulus language heavily tying the company to this tool or the requirements need to be translated to Stimulus language. Just having the requirements in DOORS using the Stimulus language is not enough but the benefits of using the syntax do not realize until the Stimulus tool is used. This translation process causes its own problems as Stimulus does not offer any automated system for this and a lot of mistakes can be made in the translation process.

In Nokia team, there was also discussion on whether the Stimulus language takes a stand on implementation. For example, in the attribute and interface names and Stimulus language in general is very code like. As discussed in chapter 2 Prior research this may be against good practices.

The results show that further examination is needed to determine whether the tool is useful for Nokia or not. The biggest hinder at the moment with Stimulus is the poor integration with Magic Draw. Dassault has promised to fix these issues in future. More examination is needed to see if the issues are fixed accordingly. At the moment because of this issue, the effort required to create and maintain the models reduces significantly the effort versus benefits gained from the tool.

The workflow also needs to be thought accordingly. As Nokia does not want to tie itself too much to one tool and because the requirements need to be read and understood by people with no knowledge of Stimulus and with only basic understanding of the system, the requirements need to be displayed in natural language in DOORS which is the main requirements management tool used. The workflow used in this study would cause problems as requirements would have to be translated twice. First, the requirements would be created in natural language then translated to Stimulus language and then back to natural language once the issues have been fixed. This increases the effort hugely and the issues that come with translation come twice in the workflow. In further examination better workflow needs to be thought out. The MBSE exercise at Nokia will go to the next phase where these issues with the workflow need to be thought through.

## References

- Abbasi, M. A., Jabeen, J., Hafeez, Y., Batool, D., & Fareen, N. (2015). Assessment of requirement elicitation tools and techniques by various parameters. *Software Engineering*, 3(2), 7-11.
- Ali, S. W., Ahmed, Q. A., & Shafi, I. (2018). Process to enhance the quality of software requirement specification document. Paper presented at the *2018 International Conference on Engineering and Emerging Technologies (ICEET)*, 1-7.
- Azuma, M. (1996). Software products evaluation system: Quality models, metrics and processes—International standards and Japanese practice. *Information and Software Technology*, 38(3), 145-154.
- Boegh, J. (2008). A new standard for quality requirements. *IEEE Software*, 25(2), 57.
- Bruel, J., Ebersold, S., Galinier, F., Naumchev, A., Mazzara, M., & Meyer, B. (2019). The role of formalism in system requirements (full version). *arXiv Preprint arXiv:1911.02564*,
- Collofello, J. S. (1988). Introduction to software verification and validation. *Introduction to Software Verification and Validation*,
- Cross, N. (1993). A history of design methodology. *Design Methodology and Relationships with Science*, , 15-27.
- Cross, N. (2001). Designerly ways of knowing: Design discipline versus design science. *Design Issues*, 17(3), 49-55.
- Dassault Systèmes. (2021). *STIMULUS – Requirement simulation – CATIA - Dassault Systèmes*. <https://www.3ds.com/products-services/catia/products/stimulus/>

- de Gea, Juan M Carrillo, Nicolás, J., Alemán, J. L. F., Toval, A., Ebert, C., & Vizcaíno, A. (2011). Requirements engineering tools. *IEEE Software*, 28(4), 86-91.
- Fabbrini, F., Fusani, M., Gnesi, S., & Lami, G. (2000). Quality evaluation of software requirement specifications. Paper presented at the *Proceedings of the Software and Internet Quality Week 2000 Conference*, 1-18.
- Fanmuy, G., Fraga, A., & Llorens, J. (2012). Requirements verification in the industry. *Complex systems design & management* (pp. 145-160) Springer.
- Fischer, K. F., & Walker, M. G. (1979). Improved software reliability through requirements verification. *IEEE Transactions on Reliability*, 28(3), 233-240.
- Fuentes, J., Fraga, A., Génova, G., Parra, E., Alvarez, J. M., & Llorens, J. (2016). Applying INCOSE rules for writing high-quality requirements in industry. Paper presented at the *INCOSE International Symposium*, , 26(1) 1875-1889.
- Hevner, A. R. (2007). A three cycle view of design science research. *Scandinavian Journal of Information Systems*, 19(2), 4.
- Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research 1. *MIS Quarterly*, 28(1), 75-105. Retrieved from <https://www.proquest.com/scholarly-journals/design-science-information-systems-research-1/docview/218119584/se-2?accountid=13031>
- Huertas, C., & Juárez-Ramírez, R. (2012). NLARE, a natural language processing tool for automatic requirements evaluation. Paper presented at the *Proceedings of the CUBE International Information Technology Conference*, 371-378.
- IBM Corporation. (2021). *Overview of DOORS Next - IBM Documentation*. <https://www.ibm.com/docs/en/elm/7.0.3?topic=capabilities-doors-next>

- J. Bubenko, C. Rolland, P. Loucopoulos, & V. DeAntonellis. (1994). Facilitating "fuzzy to formal" requirements modelling. Paper presented at the - *Proceedings of IEEE International Conference on Requirements Engineering*, 154-157.  
doi:10.1109/ICRE.1994.292391
- Joshi, S. D., & Deshpande, D. (2012). Textual requirement analysis for UML diagram extraction by using NLP. *International Journal of Computer Applications*, 50(8), 42-46.
- Kelanti, M. (2016). *Stakeholder analysis in software-intensive systems development*  
Retrieved from <https://oula.finna.fi/Record/oy.9915207173906252>
- Lee, B., Hwang, M., Lee, Y., Lee, H., Baik, J., & Lee, C. (2009). Design and development of a standard guidance for software requirement specification. *Journal of KIISE: Software and Applications*, 36(7), 531-538.
- Lutz, R. R. (1996). Targeting safety-related errors during software requirements analysis. *Journal of Systems and Software*, 34(3), 223-230.
- Nazir, F., Butt, W. H., Anwar, M. W., & Khattak, M. A. K. (2017). The applications of natural language processing (NLP) for software requirement engineering-a systematic literature review. Paper presented at the *International Conference on Information Science and Applications*, 485-493.
- Osman, M. H., & Zaharin, M. F. (2018). Ambiguous software requirement specification detection: An automated approach. Paper presented at the *2018 IEEE/ACM 5th International Workshop on Requirements Engineering and Testing (RET)*, 33-40.
- Pace, D. K. (2004). Modeling and simulation verification and validation challenges. *Johns Hopkins APL Technical Digest*, 25(2), 163-172.

- Pfeffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2006). The design science research process: A model for producing and presenting information systems research. Paper presented at the *Proceedings of the First International Conference on Design Science Research in Information Systems and Technology (DESRIST 2006), Claremont, CA, USA*, 83-106.
- Powell, P. B. (1982). Software validation, verification, and testing technique and tool reference guide.
- Ramesh, B. (1998). Factors influencing requirements traceability practice. *Communications of the ACM*, 41(12), 37-44.
- Ramesh, B., Powers, T., Stubbs, C., & Edwards, M. (1995). Implementing requirements traceability: A case study. Paper presented at the *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*, 89-95.
- Ryan, M. J., Wheatcraft, L. S., Dick, J., & Zinni, R. (2014). An improved taxonomy for definitions associated with a requirement expression. *Systems Engineering and Test and Evaluation*, 28 Apr-30,
- Schneider, F., & Berenbach, B. (2013). A literature survey on international standards for systems requirements engineering. *Procedia Computer Science*, 16, 796-805.
- Shah, U. S., & Jinwala, D. C. (2015). Resolving ambiguities in natural language software requirements: A comprehensive survey. *ACM SIGSOFT Software Engineering Notes*, 40(5), 1-7.
- Sharma, Y., & Sharma, A. K. (2014). Evaluation of the software requirement tools. *International Journal of Engineering Research*, 3(3)

- Singh, M. (2018). Automated validation of requirement reviews: A machine learning approach. Paper presented at the *2018 IEEE 26th International Requirements Engineering Conference (RE)*, 460-465.
- Walia, G. S., & Carver, J. C. (2009). A systematic literature review to identify and classify software requirement errors. *Information and Software Technology*, *51*(7), 1087-1109.
- Wautelet, Y., Heng, S., Kolp, M., & Mirbel, I. (2014). Unifying and extending user story models. Paper presented at the *International Conference on Advanced Information Systems Engineering*, 211-225.
- Wiels, V., Delmas, R., Doose, D., Garoche, P., Cazin, J., & Durrieu, G. (2012). Formal verification of critical aerospace software. *AerospaceLab*, (4), p. 1-8.



## Appendix A. Requirements

#	Name	Text
1	4477383 Clock class can only get values from the given list	Clock class can only get values from the given list Valid list of Clock class = [6, 7, 135, 140, 150, 160, 165, 248]
2	4477384 Domain number needs to be equal or between 44 and 63	Domain number needs to be equal or between 44 and 63 $43 < \text{Domain number} < 64$
3	4477387 PTP Time needs to be traceable	PTP Time needs to be traceable
4	4477388 Current UTC offset received in Announce Message must be valid	Current UTC offset received in Announce Message must be valid
5	4477390 PTP tuning profile for non-compliant networks	Configurable PTP tuning profile for non-compliant networks
6	4477403 Network two-way packet delay must not exceed 125us	Network two-way packet delay must not exceed 125us (pktSelected2WayTE metric from ITU-T G.8260 clause I.3.2.2)
7	4477404 Alarm must be raised if phase and time synchronization threshold is exceeded	Alarm must be raised if phase and time synchronization threshold is exceeded
8	4477405 Filtering time for raising the alarm shall be 5s	Filtering time for raising the alarm shall be 5s

#	Name	Text
9	4477578 PTP Time Scale must be true	PTP Time Scale must be true Parameter PTP Time Scale = True
10	4477581 Frequency needs to be traceable	Frequency needs to be traceable Set with parameter Frequency Traceable = True
11	4477925 Configurable synchronization mode support	Configurable synchronization mode support
12	4477926 Synchronization mode support for network level time and phase synchronization.	Synchronization mode support for network level time and phase synchronization.
13	4477927 PTP tuning profile for compliant network	Configurable PTP tuning profile for loose G.8275.2 support on G.8265.1 networks
14	4477928 PTP tuning profile for loose phase	Configurable PTP tuning profile for loose phase
15	4477930 Loose PTP time and phase synchronization threshold absolute value is 250us	Loose PTP time and phase synchronization threshold absolute value is 250us
16	4477931 Filtering time for clearing the alarm shall be 5s	Filtering time for clearing the alarm shall be 5s

#	Name	Text
17	4478036 Synchronization reference source needs to support PTP 1588v2-2019	Synchronization reference source needs to support PTP 1588v2-2019
18	4478155 PTP estimator optimization for 250us loose PTP synchronization solution.	PTP estimator optimization for 250us loose PTP synchronization solution.
19	4478156 PTP jump detector shall be disabled for PTP loose profile	PTP jump detector shall be disabled for PTP loose profile
20	4478754 Support for PTP protocol according to IEEE 1588v2.1-2019	Support for PTP protocol according to IEEE 1588v2.1-2019
21	4478755 Support for PTP slave functionality according to ITU-T G.8265.1 frequency standard profile	Support for PTP slave functionality according to ITU-T G.8265.1 frequency standard profile
22	4478756 Support for PTP slave functionality according to ITU-T G.8275.1 multicast standard profile	Support for PTP slave functionality according to ITU-T G.8275.1 multicast standard profile
23	4478757 Support for PTP slave functionality according to ITU-T G.8275.2 unicast standard profile	Support for PTP slave functionality according to ITU-T G.8275.2 unicast standard profile
24	4478758 Phase synchronization mode must have 120s tuning cycle length with PTP as synchronization reference	Phase synchronization mode must have 120s tuning cycle length with PTP as synchronization reference

#	Name	Text
25	4478759 Initial coarse tuning cycle length must be parametrized	Initial coarse tuning cycle length must be parametrized to following list [12s, 60s]
26	4478760 Coarse tuning packet selection window max length must be 1 tuning cycle with Normal PTP profile	Coarse tuning packet selection window max length must be 1 tuning cycle with Normal PTP profile
27	4478761 Coarse tuning packet selection window max length must be 10 tuning cycles with Custom PTP profile	Coarse tuning packet selection window max length must be 10 tuning cycles with Custom PTP profile
28	4478762 Normal tuning packet selection window must be 5 tuning cycles	Normal tuning packet selection window must be 5 tuning cycles
29	4478763 Provide BTS system clock phase error and frequency deviation against external synchronization reference source	Provide BTS system clock phase error and frequency deviation against external synchronization reference source
30	4478764 Packet counters for received and sent PTP messages must be provided	Packet counters for received and sent PTP messages must be provided
31	4478765 PTP slave packet selection window must be flushed and restarted when common BCN is set with other reference source	PTP slave packet selection window must be flushed and restarted when common BCN is set with other reference source
32	4478766 Packet selection window timestamps must be rescaled based on tuning frequency change	Packet selection window timestamps must be rescaled based on tuning frequency change

#	Name	Text
33	4478767 Availability status must be provided for all configured masters	Availability status must be provided for all configured masters
34	4478768 PTP Slave must monitor availability of all configured PTP masters	PTP Slave must monitor availability of all configured PTP masters based on received PTP messages
35	4478769 PTP Slave must not set configured PTP master available until messages are received	PTP Slave must set configured PTP master available only when Announce message, Sync message and Delay_resp messages are received from PTP Master
36	4478770 PTP slave must monitor announce messages	PTP slave must validate and save status parameters from received announce messages
37	4478771 PTP slave must support PTP masters modes	PTP slave must support PTP masters using one-step and two-step mode
38	4478772 PTP slave must automatically switch to the PTP master mode	PTP slave must be able to automatically adopt mode used by the PTP master
39	4478773 Configured PTP master must be usable only under certain conditions	Configured PTP master must be usable only when received clock class, domain number and sync message rate are within configured ranges
40	4478774 BTS must be able to detect phase jumps in PTP timing packets	BTS must be able to detect phase jumps in PTP timing packets

#	Name	Text
41	4478775 BTS must tolerate phase jumps in PTP network	BTS must tolerate phase jumps larger than 2.8us in PTP network
42	4478776 PTP slave must be unavailable when phase jump is detected	PTP slave must be unavailable for 6 tuning cycles when phase jump is detected
43	4478777 PTP slave must restart counting tuning cycles if another phase jump happens during unavailability	PTP slave must restart counting of 6 tuning cycles if during unavailability there is detected another phase jump larger than 2.8us
44	4478778 Reference source selection must be based on configured priority and reference usability and availability	Reference source selection must be based on configured priority and reference usability and availability
45	4478779 Reference source without configured priority must not be selected as reference source	Reference source without configured priority must not be selected as reference source
46	4478780 BTS must start holdover algorithm in case of no reference source available and usable	BTS must start holdover algorithm in case of no reference source available and usable
47	4478781 PTP slave must be time source when it is selected as reference source	PTP slave must be time source when it is selected as reference source
48	4498918 Allowed message rates [1, 16, 32, 64, 128] pkt/s	Allowed message rates [1, 16, 32, 64, 128] pkt/s

#	Name	Text
49	4498927 Measurement period [100-200s]	Measurement period [100-200s]
50	4498958 Total number of the received sync messages (message rate * measurement time) must meet configured message rate within measurement time (measurement period + 10% tolerance)	Total number of the received sync messages (message rate * measurement time) must meet configured message rate within measurement time (measurement period + 10% tolerance)
51	4498989 Announce message must be received at least once within 5s	Announce message must be received at least once within 5s
52	4498996 Sync message must be received at least in period max[0.5s; time for 4 packets with configured Message Rate]	Sync message must be received at least in period max[0.5s; time for 4 packets with configured Message Rate]
53	4498999 Delay_respmessage must be received at least in period max[0.5s; time for 4 packets with configured Message Rate]	Delay_respmessage must be received at least in period max[0.5s; time for 4 packets with configured Message Rate]
54	4499004 Received clock class in announce messages must match with configured (allowed) clock classes	Received clock class in announce messages must match with configured (allowed) clock classes
55	4499009 Received Domain Number must be same as configured Domain Number in all messages (announce, sync, delay_resp)	Received Domain Number must be same as configured Domain Number in all messages (announce, sync, delay_resp)
56	4499016 Fault must be raised about "Missing PTP reference" if source remain "Unavailable" or "Unusable" over 5min	Fault must be raised about "Missing PTP reference" if source remain "Unavailable" or "Unusable" over 5min

#	Name	Text
57	4499024 "Missing PTP reference" fault must be cleared immediately when source become "Available" and "Usable" again	"Missing PTP reference" fault must be cleared immediately when source become "Available" and "Usable" again
58	PTP availability must have 5s unavailable available timer	PTP availability must have 5s losing timer in between available and unavailable states to avoid too short state transition between main states (unavailable/available)
59	PTP availability states 15s restore timer	PTP availability must have 15s wait to restore timer between unavailable and available to avoid too short state transition between main states (unavailable/available)