

Eero Saarinen

**IMPROVING IEC 61131-3 SEQUENTIAL
FUNCTION CHART STANDARD TO FIT
MODERN AUTOMATION NEEDS**

Master of Science Thesis

ABSTRACT

Eero Saarinen: Improving IEC 61131-3 sequential function chart standard to fit modern automation needs

Tampere University

Master of Science Thesis, 53 pages, 4 Appendix pages

March 2019

Master's Degree Programme in Automation Technology

Major: Information Systems in Automation

Examiner: Professor Matti Vilkkö

Keywords: IEC 61131-3, SFC, sequence

IEC 61131-3 standard defines the programming languages that are used to program programmable logic controllers. Among those languages is Sequential Function Chart, that can be used to configure sequential controls. Sequences can be used to control anything that needs to be executed sequentially one step at a time. Some example use cases for sequential control are washing machine washing cycle, controlling traffic lights or transitioning a machinery from on to off.

In this thesis, the goal was to find how the standard could be improved to better fit the cases it is used for. The main research method was interviewing engineers that have some experience in configuring sequences. The interviews were conducted as semi-structured interviews to get the most out of the users of the tools that implement the standard.

Some of the most important features that were missing from the standard were the lack of special error handling, a simple way to define non-Boolean actions and too permitting branch execution. There were also a lot of small implementation details that is worthwhile to consider when implementing the standard. The most important being the translation between structured text and SFC and the amount of details that are shown in the diagram without making it too cluttered.

TIIVISTELMÄ

Eero Saarinen: IEC 61131-3 sekvenssidiagrammi-standardin parantaminen nykyajan automaation tarpeisiin
Tampereen yliopisto
Diplomityö, 53 sivua, 4 liitesivua
Maaliskuu 2019
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Automaation tietotekniikka
Tarkastaja: professori Matti Vilkkö

Avainsanat: IEC 61131-3, SFC, sekvenssi

IEC 61131-3 standardi määrittelee ohjelmoitavien logiikoiden ohjelmointikieliä. Näiden joukossa on Sequential Function Chart -kieli, jota käytetään sekvenssien määrittelyyn. Sekvenssejä käytetään, kun laitteistoa tulee ohjata vaihe kerrallaan. Tällaisia toimintoja ovat esimerkiksi pyykinpesukoneen pesuohjelmat, liikennevalojen ohjaus ja tuotantolaitteiston käynnistäminen tai sammuttaminen.

Tässä työssä tavoitteena oli kartoittaa standardiin parannuksia, jotka parantavat sen soveltuvuutta sen käyttökohteisiin. Tutkimus suoritettiin pääasiassa haastattelemalla insinöörejä, joilla on kokemusta sekvenssien ohjelmoinnista. Parhaan tuloksen saamiseksi haastattelut pidettiin temahaastatteluina.

Erillisen virhetilanteiden käsittelyn puute, liian suppea tuki sekvenssin toimintojen määrittelylle ja liian salliva haaroitus nähtiin standardin suurimpina heikkouksina. Lisäksi työssä nousi esiin toteutukseen liittyviä yksityiskohtia, jotka tulisi ottaa huomioon standardia toteuttaessa. Tärkeimmät näistä olivat Structured Textin ja Sequential Function Chartin välinen käännös ja kaaviossa näytettävien yksityiskohtien liiallisen määrän välttäminen.

PREFACE

This thesis was made during 2018 for Valmet Automation Oy Research and Development department. I've been previously working there as a trainee for two summers plus half a year.

This thesis was examined by Professor Matti Vilkkko from Tampere University of Technology with the help of Professor Hannu Koivisto. I want to thank them both for their excellent advises in the starting phases of writing this thesis. And for guiding me to the right track when trying to figure out which research method to use.

From Valmet, the thesis was supervised by Ari Leppäniemi, the manager of our team. I want to thank him for giving me enough slack to be able to finish writing this thesis in time. Big thanks for that also goes to Jari Tamminen, who told me that he will ensure that I'll get enough time to write my thesis in time.

I also want to thank our whole team which gave me support and made coming to work a joy all the way throughout the writing process. Especially I want to thank Nina Flink for including me in interviewing users so that I got a little experience for making interviews on my own.

Special thanks also belong to Veera Valutie for supporting me and keeping me motivated all the way throughout the process.

Tampere, 14.3.2019

Eero Saarinen

CONTENTS

1	INTRODUCTION	1
2	BACKGROUND	4
	2.1 IEC 61131-3	5
	2.1.1 Overview	6
	2.1.2 Instruction List (IL).....	6
	2.1.3 Structured Text (ST)	7
	2.1.4 Function Block Diagram (FBD)	9
	2.1.5 Ladder Diagram (LD)	10
	2.1.6 Sequential function chart	11
	2.2 GRAFCET.....	16
	2.3 Existing SFC tools.....	17
	2.3.1 DNA Sequence CAD	17
	2.3.2 Siemens PCS.....	19
	2.3.3 ABB 800xA system	20
	2.3.4 CODESYS	21
3	RESEARCH METHODS	24
	3.1 Qualitative research methods	24
	3.1.1 Structured interview	25
	3.1.2 Unstructured interview.....	26
	3.1.3 Semi-structured interview	26
	3.2 Exploring literacy.....	28
	3.3 Benchmarking SFC tools	29
4	END-USER INTERVIEWS	31
	4.1 Interviewed users.....	31
	4.2 Interview topics	32
	4.2.1 Steps.....	33
	4.2.2 Branches.....	33
	4.2.3 Syntax verification	34
	4.2.4 Using structured text	35
	4.2.5 Templating	35
	4.2.6 Other.....	36
	4.3 Conclusion.....	36
5	RESULTS	38
	5.1 Steps	39
	5.2 Transitions.....	41
	5.3 Branching	42
	5.4 Testing.....	43
	5.5 Structured text	44
	5.6 Templates	45
	5.7 Results from other sources	46

5.8	Discussion	46
6	CONCLUSIONS.....	49
7	REFERENCES.....	51

APPENDIX A: EXAMPLE SFC WRITTEN IN ST

APPENDIX B: INTERVIEW TOPICS AND SUPPORTING QUESTIONS

LIST OF SYMBOLS AND ABBREVIATIONS

SFC	Sequential Function Chart
DCS	Distributed Control System
IEC	International Electrotechnical Commission
PLC	Programable Logic Controller
IL	Instruction List
ST	Structured Text
LD	Ladder Diagram
FBD	Function Block Diagram
ISA	International Society of Automation
FDA	Food and Drugs Administration

1 INTRODUCTION

IEC 61131-3 (International Electrotechnical Commission) standard defines 5 different ways to program PLC (Programmable Logic Controller). It defines two text based programming languages: instruction lists (IL) and structured text (ST) and three graphical languages: ladder diagram (LD), function block diagram (FBD) and sequential function chart (SFC) (John & Tiegelkamp, 2010, p. 99; IEC, 2013). In this thesis, the focus is on sequential function charts. Other programming languages will also be discussed briefly, as they closely relate to each other. The main sources for this thesis are the IEC standard (2013) and textbooks that explain the standard in more detail, such as John and Tiegelkamps textbook on the standard (2010).

Sequential function charts together with all the other IEC 61131-3 languages are needed to implement the automation system from process descriptions into working control applications. Sequential function charts are used to graphically define a sequence of events that are involved in systems operation (Bolton, 2009). A sequence could control anything from starting up a ships engine to traffic lights. Typical use case for SFC programming is a chemical process that requires filling, mixing and emptying containers at the right time and often in parallel (John & Tiegelkamp, 2010, p. 169).

Sequence CAD - the current sequencing tool in use in Valmet DNA - is implemented in the 1990's and has not received any big remakes since then. Sequence CAD is loosely based on the IEC SC65A 67-I DRAFT that later became the IEC 61131-3 standard. It is, however lacking some features like branching and thus, is not fully IEC 61131-3 compliant. There are also many features that are not fully in compliance with the standard. (Valmet Automation Oy, 2017; IEC, 2013)

There are already existing extensions of the IEC 61131-3 standard like the one maintained by PLCopen. They are a worldwide non-profit organization that provides training, certification and benchmarking of IEC 61131-3 compliant tools among other things. They have published extensions to the standard and gives guidelines for programming PLCs. (Lydon, 2012; PLCopen, 2013)

This thesis builds on the question: how could IEC 61131-3 SFC standard be improved to suit modern day needs? The goal is to determine what key features are missing from current standard and what are some essential features that are must haves in any SFC designer tool.

The standard states many ways of defining sequences. SFC may contain FBD, IL, ST or LD as conditions for steps (John & Tiegelkamp, 2010). Many ways on doing the same thing is in many ways an asset, because it allows using the right tool for the right job. However, it can result in some bad and hard to maintain configurations if the wrong type of tool is used. It also can make the tool hard to use for new users.

The IEC 61131-3 standard does not define a way of utilizing templates. Templates would allow to reuse similar code in many places without the need for rewriting the whole code for each sequence. (Fuchs, et al., 2014). Templating would make configuring a big plant very cost-effective and easy.

The standard also lacks in static verification of the produced code (Fujino, et al., 2000; Fuchs, et al., 2014; Okuda, et al., 2013). The main method of SFC debugging is by executing it and verifying its functionality in runtime, this is also known as dynamic analysis (Okuda, et al., 2013).

Most probably IEC 61131-3 standard is also lacking efficiency in batch process needs. Recipe controlling is usually achieved by very complex and often hard to maintain control code (Ferrarini & Piroddi, 2002). In batch processes one recipe might have to be executed in several different plants with different equipment (Smith, 2014, p. 295). One solution to this problem might be a support for templating in the SFC tool.

A possible outcome might also be, that users need a method to jump from one step to another mid-sequence. The sequence execution will probably also need a way to safely shut down or pause the execution. Pausing the process execution with all valves open, might be a serious safety issue.

The goal is to identify problems in IEC 61131-3 standard by interviewing different kind of users. The preferred outcome would be to have as much improvements and notes as possible. It's important to base the research on as diverse user group as possible to ensure the product suits all kind of needs. To help the interviews, a few different existing implementations should be explored to demonstrate different kind of approaches. This thesis will serve as a preliminary study on how Valmet DNA sequence configuration tool, Sequence CAD, could be improved to better suit the user's needs. Additionally, this thesis will help the developers to better understand the IEC 61131-3 SFC standard.

The main research method used in this thesis is semi-structured interviews. Semi structured interviews are flexible enough to make it possible for new ideas to emerge. They are also not as demanding for an inexperienced interviewee as unstructured interviews. In addition, related research and literature is explored so the interviews can be used to verify the findings from those.

In this thesis, the IEC 61131-3 standard is first briefly introduced in chapter 2.1 to familiarize the reader to the basic concepts and more importantly to the sequential function chart. Some existing sequence configuration tools are introduced and discussed in chapter 2.3 to show how the standard is implemented in those. Information about different SFC tools are mainly looked up from their user manuals or by testing the tool, if possible. In chapter 3, the research methods used in this thesis are introduced and in chapter 4 they are discussed and applied to fit the research. The results are shown in chapter 5 separated into different parts to the standard they apply.

2 BACKGROUND

In this chapter, we'll go through what is sequential control and describe some of the techniques that are used to configure sequential control. Other programming languages of the IEC 61131-3 standard are also introduced as they are a vital part of programming SFCs. We'll also explore some of the existing sequence configuration tools from different major automation system providers such as Valmet, Siemens and ABB.

Sequences are mostly used to describe a sequential behaviour of a process. Sequences are also used to separate complex logic to smaller and simpler blocks, that can manage one part of the process without concern of the other parts of the process. (Lewis, 1998, p. 186)

Sequential control is much needed in automated batch processes as they have clearly defined phases that need to be executed sequentially. Continuous processes also require sequential control as any continuous process has some identifiable states, such as running state and shutdown state. Sequential control is useful when transitioning between these states. (Huffman, 2015)

Using sequential control can greatly improve continuous process safety. Automating the transition from running state to shutting down will reduce human error, as the shutdown procedure will always be executed the same way. The sequence can also be wired to start when a predefined alarm is triggered. That will reduce human errors and helps to prevent alarm overload. Additionally, reaction times to critical errors will be much faster, improving the safety even further. (Huffman, 2015)

To achieve sequential control capabilities, the SFC tool needs a way to describe basic arithmetic and combinatory logic. They are needed for building a set of conditions which tells the sequence when to move to the next step. The sequence also needs to access the inputs and outputs of the automation system. Inputs are needed for executing actions by writing values to the devices. Outputs are needed for reading the transition conditions from the system (Smith, 2014, p. 266)

A key feature of sequential control is to be able to suspend the execution and wait until some defined condition is satisfied. This condition can be anything that will produce a Boolean value. Typical examples include waiting for some timer or waiting for a measurement to reach some threshold. (Smith, 2014, pp. 266-267)

The first language for configuring sequences that was widely adopted was GRAFCET. It was developed by A French company called Telemecanique (John & Tiegelkamp, 2010, p. 169). GRAFCET served as a base for IEC 848 standard which has been used as a base for the IEC 61131-3 standards SFC language (John & Tiegelkamp, 2010, p. 169). The

IEC 61131-3 SFC and GRAFCET borrows a lot of its methodology from Petri-nets (John & Tiegelkamp, 2010; Johnsson & Årzén, 1999).

2.1 IEC 61131-3

The International Electrotechnical Commission or IEC is a worldwide organization that focuses on preparing and publishing standards for all technologies related to electronics and electricity. IEC consists of experts and delegates appointed by national committees of member countries. (IEC, 2018)

IEC first published the IEC 61131 standard for programmable controllers in 1993 (IEC, 2018). It provides a collection of standards to be used with PLCs. The third part of IEC 61131 defines a set of programming languages to be used for programming PLCs. The standard is composed from nine parts listed in table 1.

Table 1. *The parts of IEC 61131 standard*

Part 1	General information
Part 2	Equipment requirements and tests
Part 3	Programming languages
Part 4	User guidelines
Part 5	Communications
Part 6	Safety-related PLC
Part 7	Fuzzy control language
Part 8	Guidelines for the application and implementation of programming languages for PLC
Part 9	Single-drop digital communication interface for small sensors and actuators

The standard will be extended to include a 10th part in 2019. It will define XML format for saving programs written in any of the programming languages of part 3 of the stand-

ard. The goal is to have better transferability of programs between different PLC programming tools. The standard is based on the PLCopen XML specifications. (PLCopen, 2013)

Part 3 is the most relevant in regards of this thesis, as it contains the definition of SFC language. It defines the programming languages that are used to program PLCs. The latest version of the IEC 61131-3 standard is the third edition, published in 2003.

2.1.1 Overview

The goal for IEC 61131-3 is to have consistency in the ways that PLCs are programmed. This makes it easier to apply a control program for different PLCs across manufacturers. The language standard was built in cooperation with some of the biggest PLC companies to gain higher adoption rate for the standard. This sped up the adoption to a wider use in the automation industry. (Lewis, 1998)

The standard defines five different programming languages – two text-based and three graphical. The text based languages include instruction list (IL) - a low level language alike assembler; and structured text (ST) – a higher level textual programming language derived from Pascal (IEC, 2013, pp. 195, 201). The graphical languages include Ladder Diagrams (LD), Function Block Diagrams (FBD) and Sequential Function Charts (SFC) (IEC, 2013, p. 208). Although this thesis mainly focuses on the SFCs, understanding the basic principles of SFCs requires understanding the other programming languages of the standard as the languages are highly interconnected.

2.1.2 Instruction List (IL)

Instruction list is an assembler like language that consists of a sequence of instructions (IEC, 2013, p. 195). It can be used to define the behaviour of function block diagrams, functions or even whole programs. It can also be used in the step actions or transitions of a SFC. IL is ideal when used in very simple and straightforward problems as it is quick to write. However, solving more complex problems can prove to be difficult and time-consuming using IL. (Lewis, 1998, pp. 169-170)

```

LD      Value      (* load Measurement *)
GT      20          (* test if Value > 20 *)
JMPCD  JUMP_1      (* jump to JUMP_1 if test was false *)
LD      OtherValue (* load OtherValue *)
ADD     10          (* add 10 to OtherValue *)
ST      OtherValue (* store OtherValue *)
JUMP_1: LD      OtherValue (* load OtherValue *)
        ST      %Q42      (* store is to output 42 *)

```

Program 1. *An example of an IL program*

Program 1 is a simple example of what an IL program could look like. In the example, if some variable named *Value* is less or equal to 20, 10 is added to a variable named *OtherValue* and then *OtherValue* is written to output %Q42. Due to poor readability of assembler style languages, usually each of the instructions are commented. In IL language comments are placed between (* and *) characters. While IL might be handy in some contexts, IEC is planning to deprecate IL in the next edition of IEC 61131-3 (IEC, 2013, p. 195).

2.1.3 Structured Text (ST)

Structured text is a higher-level language, that has syntax like that of Pascal. Like IL, ST can be used to define functions and programs and it can be used in SFC step actions and transitions. ST contains all basic functionalities that is found in any other programming language of the IEC 61131-3 standard. The syntax is straightforward and therefore ST is easy to learn for anyone with some programming experience. Program 2 has the same functionality as program 1 written in ST. (Lewis, 1998, pp. 117-118)

```

IF Value > 20 THEN
    OtherValue := OtherValue + 10;
END_IF;
Output_42 := OtherValue

```

Program 2. *The example in program 1 written in ST*

As can be seen from program 2, ST is often easier to read and comprehend than IL. IEC 61131-3 also allows function block behaviour to be written in ST (Lewis, 1998, p. 145). An example of a function block can be seen in program 3. Using ST syntax to express function block could make the diagram easier to read and comprehend.

```

FUNCTION_BLOCK Example_FB
  (* inputs *)
  VAR_IN
    Input_1 : REAL;
    Input_2 : UINT;
  END_VAR

  (* outputs *)
  VAR_OUT
    Output : REAL;
  END_VAR

  (* the body *)
  IF Input_2 = 0 THEN
    Output := 0;
  ELSE
    Output := Input_1 / Input_2;
  END_IF;

END_FUNCTION_BLOCK

```

Program 3. Simple function block written in ST

Now we could use the function block like in the example seen in program 4. First, we declare a variable *Function*, that is an *Example_FB* function, now we can call the function and it will return some values.

```

VAR
  Function : Example_FB
END_VAR

Function(12.5, 10) (* returns 1.25 *)
Function(20, 0)   (* returns 0 *)

```

Program 4. Calling a function block

Structured text is the base language of the standard. Every other language can be translated to structured text (Lewis, 1998, pp. 117-118). It allows making abstractions for logic that might be hard to express with blocks, but simple to express using ST constructs. There's also some constructs that are not possible to express graphically with function blocks. For example, iteration statements are only possible by using ST (Lewis, 1998, pp. 127-130).

2.1.4 Function Block Diagram (FBD)

Function block diagrams and ladder diagrams both describe the program or a part of it with interconnected graphical blocks (IEC, 2013, p. 219). In FBD, each block represents a call to some function that has inputs and outputs (John & Tiegelkamp, 2010, pp. 140-141). A simple example function block can be seen in figure 1. (IEC, 2013; John & Tiegelkamp, 2010)

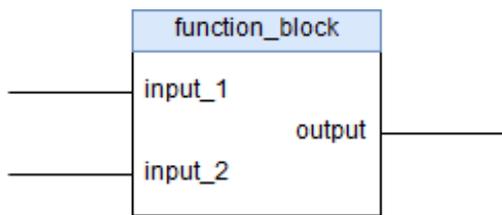


Figure 1. A function block with 2 inputs and 1 output

Function blocks are connected by connection lines. A single line can be split to multiple inputs, but multiple outputs cannot be connected to a single input. Figure 2 shows an example of interconnected function blocks. The connection lines can be connected also to special connector constructs (John & Tiegelkamp, 2010, p. 136). The connector can refer to some other part of the diagram, some other diagram or some other named value. Figure 2 contains connection blocks *connect1*, that connects output of *function_block_3* to *input_1* of *function_block_3*. Connectors are useful for dividing big function block diagram into smaller parts and to reduce the need for long wires.

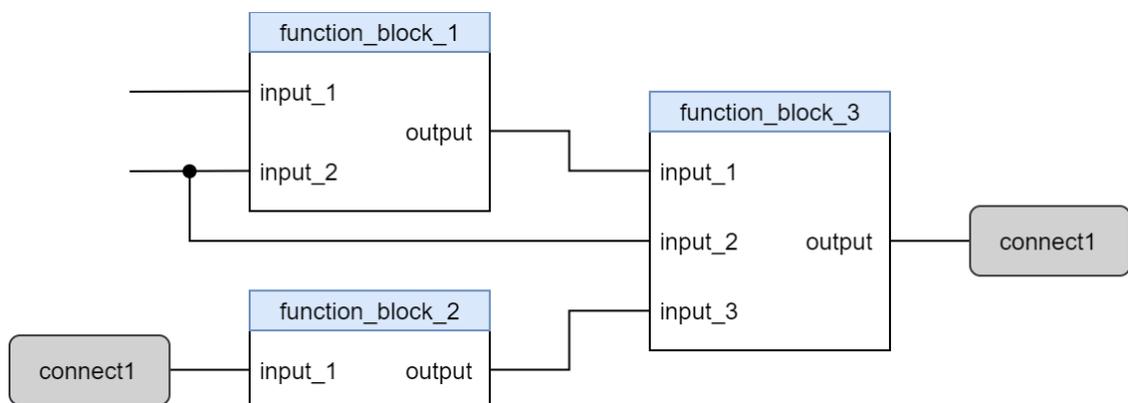


Figure 2. Interconnected function blocks with connectors

Function blocks can be translated into ST. For the most part, the translation can also be done to the opposite direction: from ST to FBD. However, some ST expressions like *WHILE* and *FOR* loops are not possible port to FBD as they don't have any graphical representation. (Lewis, 1998, p. 145)

2.1.5 Ladder Diagram (LD)

Ladder diagrams are primarily used for Boolean logic operations (John & Tiegelkamp, 2010, p. 147). The main components of LD are normally open and normally closed contacts. Boolean operators like *AND*, *OR*, *XOR* and *NAND* can be constructed by connecting these contacts. Connections in LD work the same way as in FBD with the exception that in LD multiple outputs can be connected to a single input which produces a logical *OR* (John & Tiegelkamp, 2010, p. 149). Basic Boolean operators are introduced in table 2.

Table 2. Logical operators in LD

expression	Ladder diagram expression
A	A -- --
NOT A	A -- / --
A OR B	A -- --+--- B -- --+
A XOR B	A B -- -- / --+--- A B -- / -- --+
A AND B	A B -- -- --
A NAND B	A -- / --+--- B -- / --+
B := A	A B -- --()

Ladder diagrams are useful in determining the transition condition of a SFC step as they always produce a Boolean value. Ladder diagrams can output values to variables with

coils like in the last row of **Error! Reference source not found.** (John & Tiegelkamp, 2010, p. 152; IEC, 2013, p. 218).

2.1.6 Sequential function chart

IEC 61131-3 SFC builds on top of the previously mentioned programming languages. Its building blocks - steps and transitions - can be programmed using any other IEC 61131-3 programming language (IEC, 2013, pp. 157, 160). SFC can be used to program sequential tasks like washing machine wash cycle or controlling traffic lights. SFC can also be useful for breaking down a bigger program to smaller units that are easier to maintain (John & Tiegelkamp, 2010, p. 169).

SFC is a network of steps and transitions much in the same way as FBD is a network of function blocks. The standard defines step to be a block that is configured to execute certain actions as soon as the step becomes active. Each step must have a name that can be used as its identifier. Steps have implicit values X and T. X is a Boolean value that tells whether the step is active. T represents how long the step has been active. T is reset only when the step becomes active again. (IEC, 2013, pp. 155-156)

The IEC 61131-3 standard defines that each step must be connected to a transition. Transitions keep the preceding step running until the transition condition evaluates to true. The condition can be made using any other programming language of IEC 61131-3 if it produces a Boolean value. A step is represented graphically by a block and transition by a bold line as can be seen in figure 3. Transition condition is only checked when its associated step is active and every one of the steps actions is executed at least once. Clearing the transition will deactivate all previous and activate all following steps that are connected to it. (IEC, 2013, pp. 155-159)

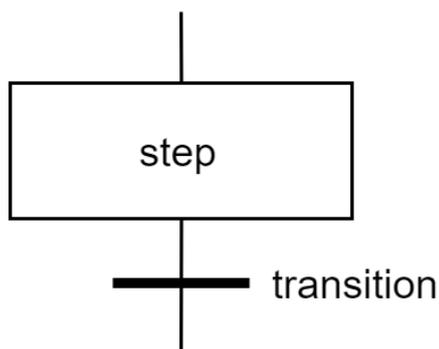


Figure 3. SFC step and transition

Each step has some actions that it will execute when it's active. A step can have zero actions, in which case the step only serves as a waiting point for its associated transition

to be fulfilled. Actions can be defined with any of the programming languages of IEC 61131-3, including SFC. Actions can be marked in the graph as shown in figure 4 or they can be omitted to keep the diagram clean. (IEC, 2013, p. 160)

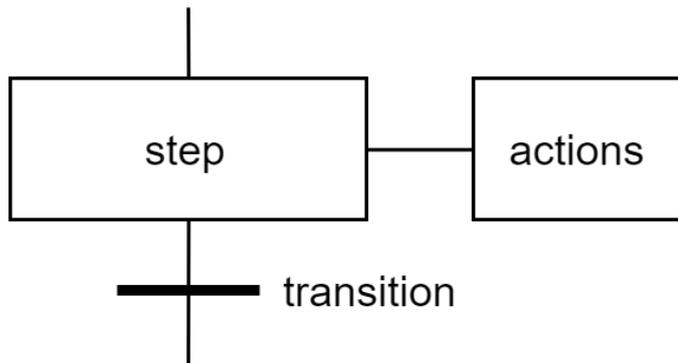


Figure 4. SFC step with actions

The actions are defined in list form where the first column represents action qualifiers. Each step should have an action qualifier that defines how or when the action should be executed. There are 11 different action qualifiers in total, denoted by one- or two-worded abbreviations (John & Tiegelkamp, 2010, pp. 193-200). These abbreviations and the action types they represent are introduced in table 3. The qualifier field can also be left empty, in which case the action is presumed to be non-stored value.

Table 3. Action qualifiers and their meanings (IEC, 2013, p. 163; John & Tiegelkamp, 2010, p. 193)

N	Non-stored value. The action is executed for as long as the step is active.
R	Overriding Reset. Reset the variable to FALSE.
S	Set. Sets the value to TRUE.
L	Time Limited. Execute the action until the time limit has reached or the step becomes deactivated.
D	Time Delayed. Execute only after the specified time has passed.
P	Pulse.

SD	Stored and time Delayed. Set to true after the specified time has passed even if the step is not active anymore.
DS	Delayed and Stored. Set to true after the specified time has passed if the step is still active
SL	Stored and time Limited. Execute action until the specified time has passed.
P1	Pulse (rising edge). Execute once after the step becomes active.
P0	Pulse (falling edge). Execute once after the step becomes deactivated.

SFC supports branches, that can be executed in parallel or conditionally (IEC, 2013, pp. 169-171). Parallel branches can be used to simultaneously execute many steps. Conditional branching can be used to have many execution options with different conditions. Conditional branches are also useful for skipping steps if a certain condition is met (John & Tiegelkamp, 2010, pp. 172-174).

Figure 5 shows an example of conditional branching, *step2* is executed only if *transition1* is fulfilled and *step3* is executed if *transition2* is fulfilled. Depending from the branch the execution is in, *step4* is executed after either *transition3* or *transition4* evaluates to true. If both *transition1* and *transition2* evaluate to true at the same time, the execution continues to *step2* since that's the leftmost branch.

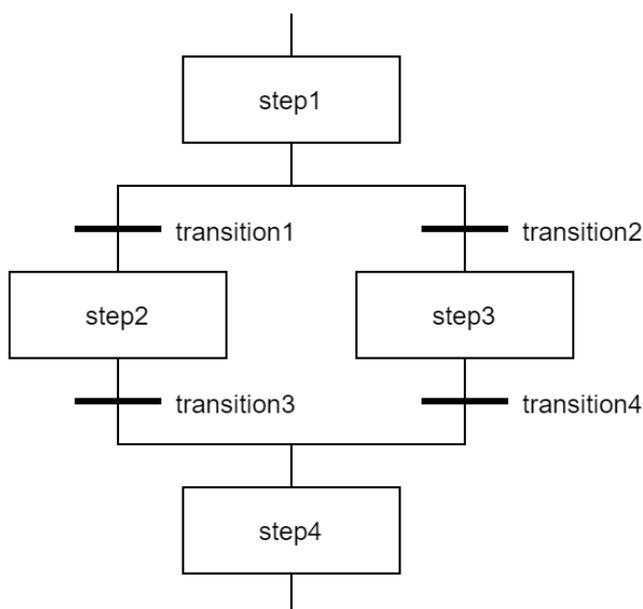


Figure 5. Sequential function chart with conditional branching

In conditional branching, only one branch may be active at the same time (IEC, 2013, pp. 169-170). When *step1* in figure 5 is active and both *transition1* and *transition2* evaluates to true, only *transition1* is cleared and the sequence moves to *step2*. By default, transition conditions are given priority from left to right (IEC, 2013, p. 169). This behaviour can be modified by numbering the branches. Then the conditions are checked in the priority given by the numbers (IEC, 2013, p. 170).

Figure 6 Shows an example of a SFC with parallel branches. As soon as *transition1* evaluates to true, both *step2* and *step3* are put into execution. They are executed consecutively until *transition2* is fulfilled and the executions moves to *step4*.

Branches can be embedded into each other, meaning that a branch could have several inner branches. Conditional branches could have inner parallel branches and vice versa. There's no restrictions on how many branches can be created from one point or how many nested branches there can be inside each other. Having many nested branches will however make the sequence hard to read and maintain. (John & Tiegelkamp, 2010)

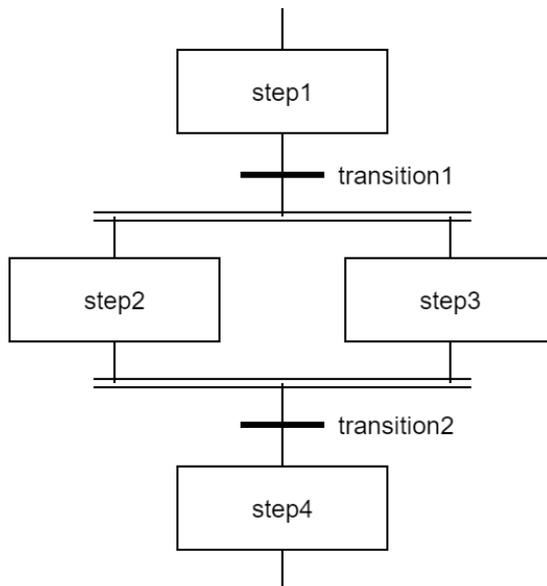


Figure 6. Sequential function chart with parallel branching

Parallel branches must end with a single transition condition and generally jumping out of a parallel execution branch is not allowed as it might result in unreachable or unsafe SFC code (John & Tiegelkamp, 2010, pp. 173-175). For example, in figure 7 if *transition5* becomes active and *step5* is executed, *step4* is never executed and *transition4* may never be fulfilled. Therefore, *step6* is never executed. On the other hand, if *transition2* is

fulfilled, the sequence may move to *step6*, but *step5* is never executed and *transition5* is never fulfilled.

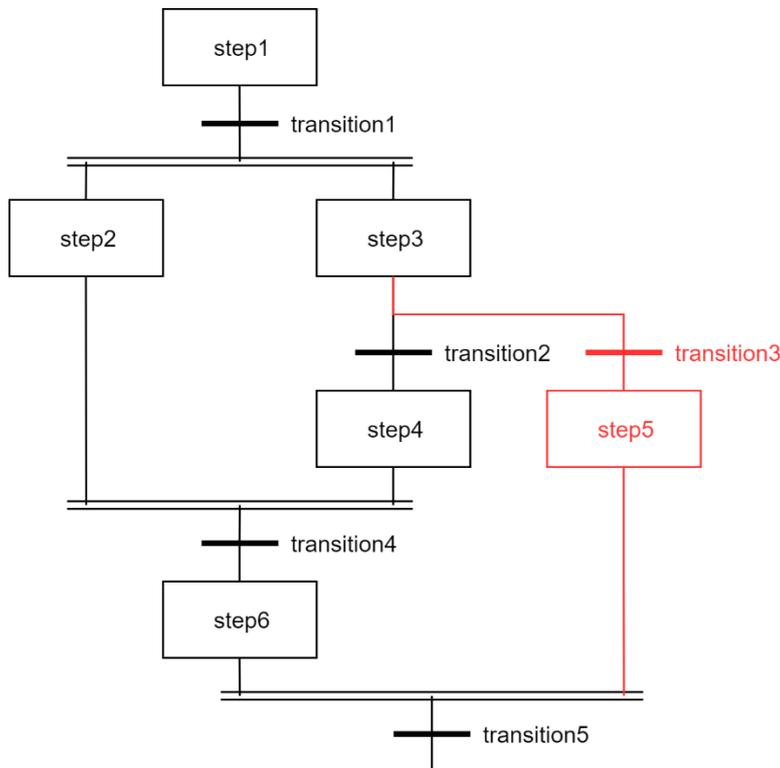


Figure 7. A SFC containing an erroneous branch emphasised in red

There are few specific cases where jumping out of parallel branch is safe and does not produce unreachable code (John & Tiegelkamp, 2010, p. 176). One example is when there are two parallel branches and there's a jump from both branches to each other. These kind of cases makes it hard to detect unsafe or erroneous sequence configurations.

Although the graphical SFC is the primary way of programming sequential controls, SFC can also be programmed textually with ST. It makes version control of SFCs simpler and enables information exchange between different systems. As an example, Program 5 is the textual presentation of the SFC program in figure 6. The ordering of the blocks is not strict, so actions might be declared either right after the steps or in the end of the program as in the example (John & Tiegelkamp, 2010, p. 204).

```

VAR
    (* All the variables are declared here *)
END_VAR

INITIAL_STEP Step1:
    Action1()
END_STEP

STEP Step2:
    (* actions of Step2 *)
END_STEP
(* etc. *)
...

TRANSITION FROM Step1 TO (Step2, Step3) := Transition1 END_TRANSITION
TRANSITION FROM (Step2, Step3) TO Step4 := Transition2 END_TRANSITION

ACTION Action1:
    (* actions executed in step1 *)
END_ACTION
(* etc. *)

```

Program 5. Example of a textual SFC

Sometimes it's easier and quicker to configure a simple sequence using the textual notation. For more complex programs, the whole program is usually easier to comprehend when presented graphically. As the textual and graphical programs are essentially the same, textual program can be transpiled into graphical one and graphical program can be transpiled into textual form.

2.2 GRAFCET

As mentioned earlier in this chapter, GRAFCET was the first widely adopted tool for programming sequences on PLCs (John & Tiegelkamp, 2010, p. 169). It was introduced in 1977 and standardized in 1982 (Baker, et al., 1987). It closely resembles the specification for Petri nets and it can be regarded as an extension of petri net (Hrúz & Zhou, 2007, p. 161).

Although GRAFCET and SFC are very similar, some differences may be noticed. The standards differ in parallel execution. SFC allows parallel execution only in a parallel branch, conditional branches only allow one branch to be executed. GRAFCET has no such limitations and multiple branches may be executed simultaneously even outside parallel branch (Baker, et al., 1987).

Sometimes the terms GRAFCET and SFC are used interchangeably. Although SFC is based on GRAFCET and for the most parts they are almost identical, it's good to keep in mind that they are not the same thing. GRAFCET standard does not specify how the

sequential control program should be implemented on the controller (Schumacher & Fay, 2011). It's merely just a specification whereas SFC is a programming language. GRAFCET needs to be transformed into a program that the controller can execute. Usually the SFC or some other IEC 61131-3 compliant language is used for that (Schumacher & Fay, 2011).

2.3 Existing SFC tools

All the biggest automation system providers have their own sequence programming tool. They are usually part of a bigger automation system configuration toolset. In this section, we consider some of the biggest and most widely used tools and compare them to the standard.

Some of the biggest actors in the automation system business include Siemens, ABB, Honeywell, Emerson and Valmet. Each of them has their own engineering tools for configuring automation system and each of those has some tool for configuring sequences. Siemens has SIMATIC, Emerson has DeltaV, Honeywell has ControlEdge and Valmet has DNA and ABB has 800xA system (ABB, 2014; Siemens AG, 2017; Valmet Automation Oy, 2017; Honeywell, 2018; Emerson, 2017).

There are also general-purpose IEC 61131-3 programming tools, that are not tied to automation vendors. Most important of them is CODESYS, developed and maintained by CODESYS group.

All the tools are based on the IEC 61131-3 standard, with varying degree of compliance to it. For the most part, the tools are stricter about the programming language to use in transitions or actions. Most of the tools are not open source or otherwise freely accessible, so the comparisons and analysis are mainly done by reading the user manuals for each tool. Emerson and Honeywell don't have user manuals publicly available for their controller configuration tools. However, both Honeywell and Emerson have IEC languages such as SFC and FBD available in their configurators (Honeywell, 2018, p. 26; Emerson, 2017).

2.3.1 DNA Sequence CAD

DNA sequence CAD is the sequence configuration tool of the Valmet DNA system. It is used as the primary tool for configuring sequential actions in automation systems delivered by Valmet. In this research, a hands-on approach could be used to investigate the DNA sequence CAD, as it is freely accessible in this research. The technical restrictions however are easier and more reliable to verify from the manuals.

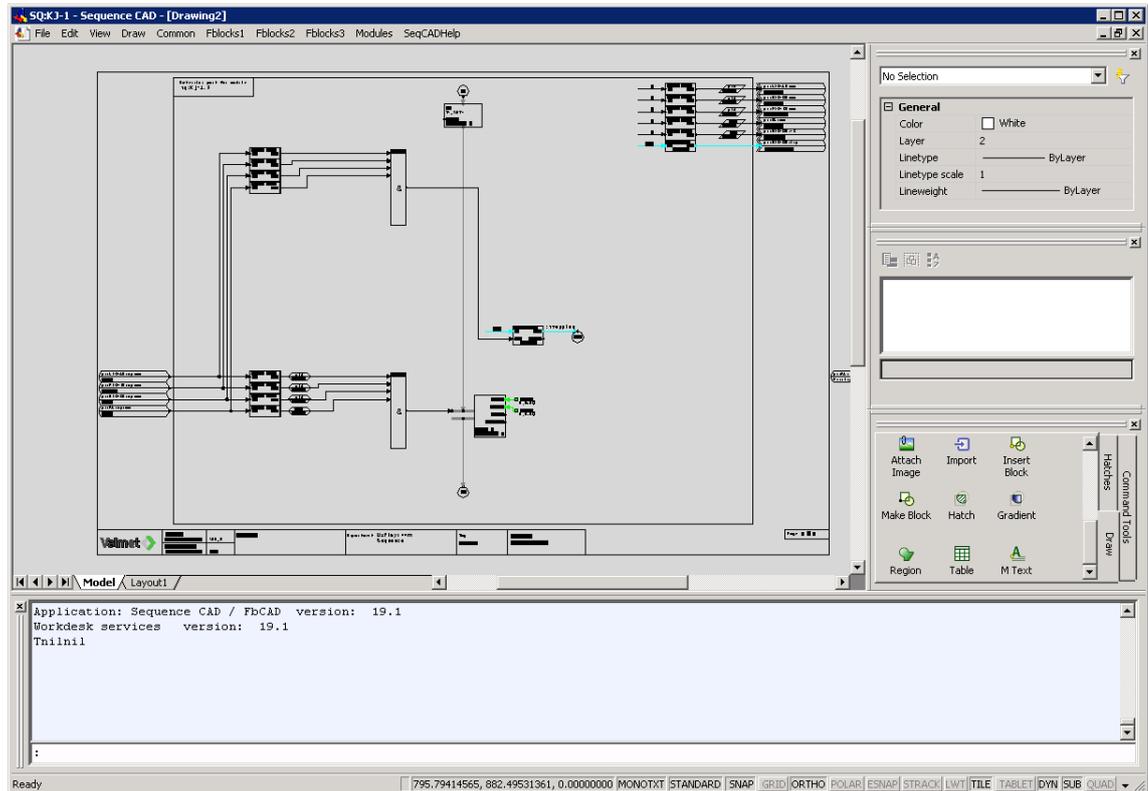


Figure 8. Valmet DNA sequence CAD

Sequence CAD is lacking some features of the standard. Per the user manuals, sequence CAD supports maximum of 52 steps per one SFC. It's not possible to create conditional or parallel branches with sequence CAD. The actions are defined by function block diagrams that are connected to some external connectors. Figure 9 shows an example of action definitions of a step in DNA sequence CAD. (Valmet Automation Oy, 2017)

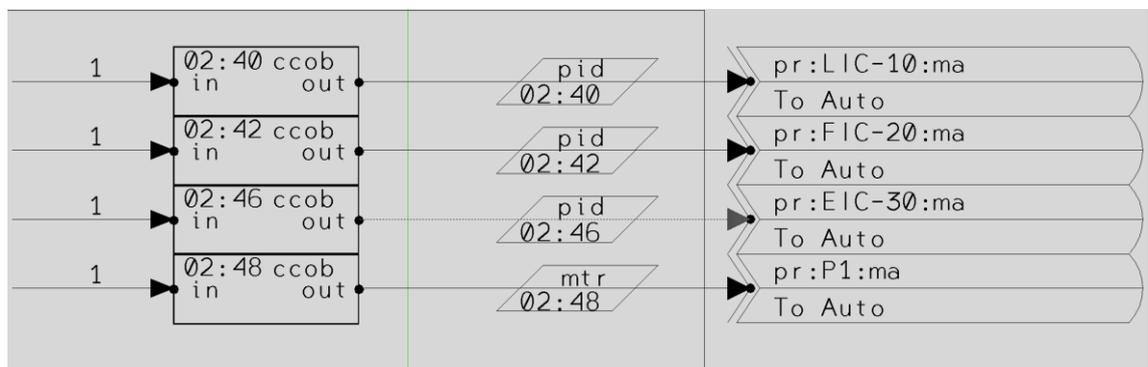


Figure 9. sequence CAD action definitions

The transitions in Valmet sequence CAD has additional behaviour compared to the standard. It's possible to define waiting time and alarm time to transition conditions (Valmet Automation Oy, 2017, p. 126). Waiting time means that the transition conditions are not tested until that time has elapsed. Alarm time means that if the transition conditions are not fulfilled within that time limit, an alarm should be raised for the operator. This is particularly useful from a security standpoint. This enables the operator to know if there's something wrong with the execution and react promptly. The alarm could also be connected to some automatic shutdown procedure.

2.3.2 Siemens PCS

Sequential control can be programmed into Siemens SIMATIC PLCs using a special programming language called GRAPH (Siemens AG, 2017, p. 7031). It is not a standard programming language and it's only used in the context of Siemens systems. An example view of sequence configuration in progress can be seen in figure 10.

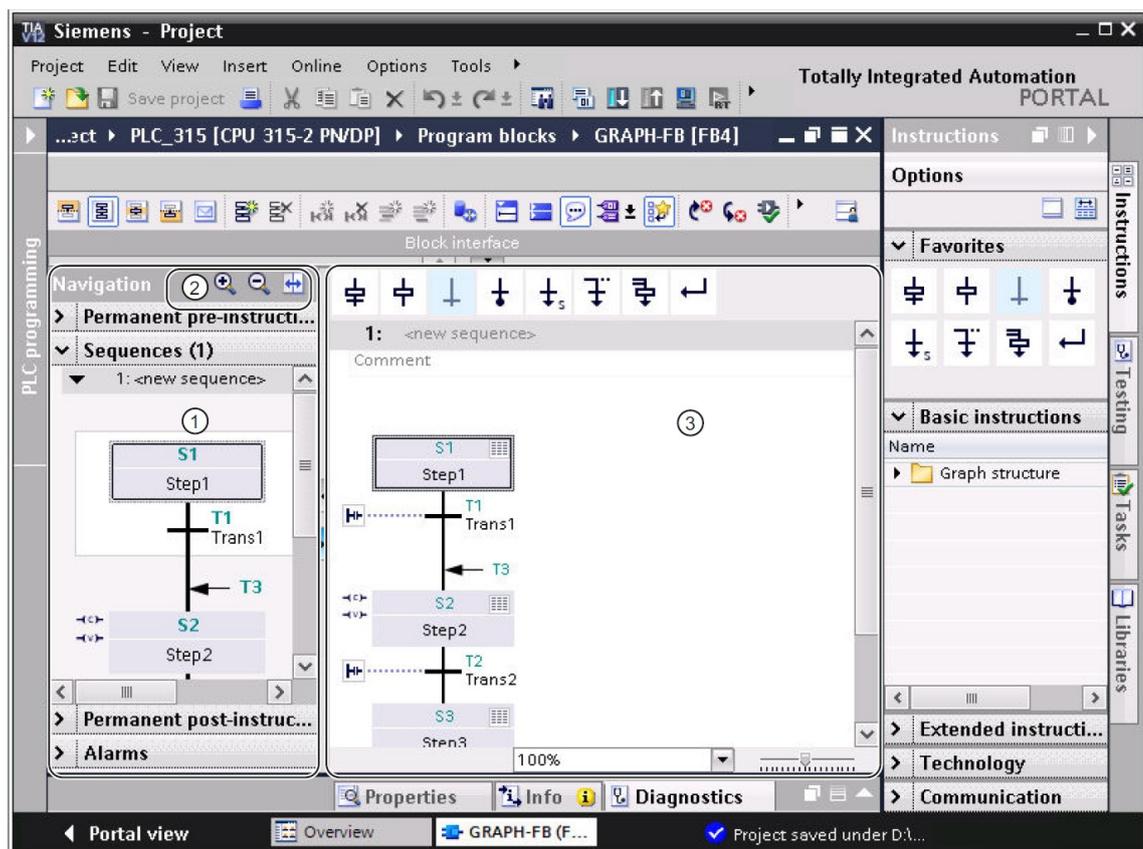


Figure 10. Siemens sequence configuration environment

The GRAPH programming language is not explicitly said to be compliant with the IEC 61131-3 standard, but it seems to have all the same functionalities and the programming logic is almost identical to that of the standard. It supports parallel and conditional branches. Additionally, GRAPH supports jumping from a step to any other step. These jumps are represented with arrows and step and transition numbering as can be seen in figure 11.

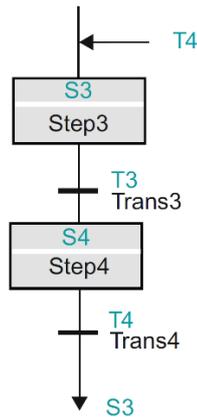


Figure 11. A jump from Trans4 to Step3 (Siemens AG, 2017)

GRAPH differs from the IEC 61131-3 standard in the way of defining step actions. It only allows the actions to be defined textually by either an operand, assignment or a function block call (Siemens AG, 2017, p. 7041). Transitions can be programmed using either LD or FBD, ST or IL is not supported (Siemens AG, 2017, p. 7051).

2.3.3 ABB 800xA system

ABB 800xA system configuration tool can be used to program SFCs. ABB 800xA doesn't have a separate sequence configuration tool, instead sequences are configured with the Function Designer (ABB, 2014, p. 62). Figure 12 is a screenshot of the sequence edition tool. We can observe that the diagram strongly resembles to the ones specified by the standard.

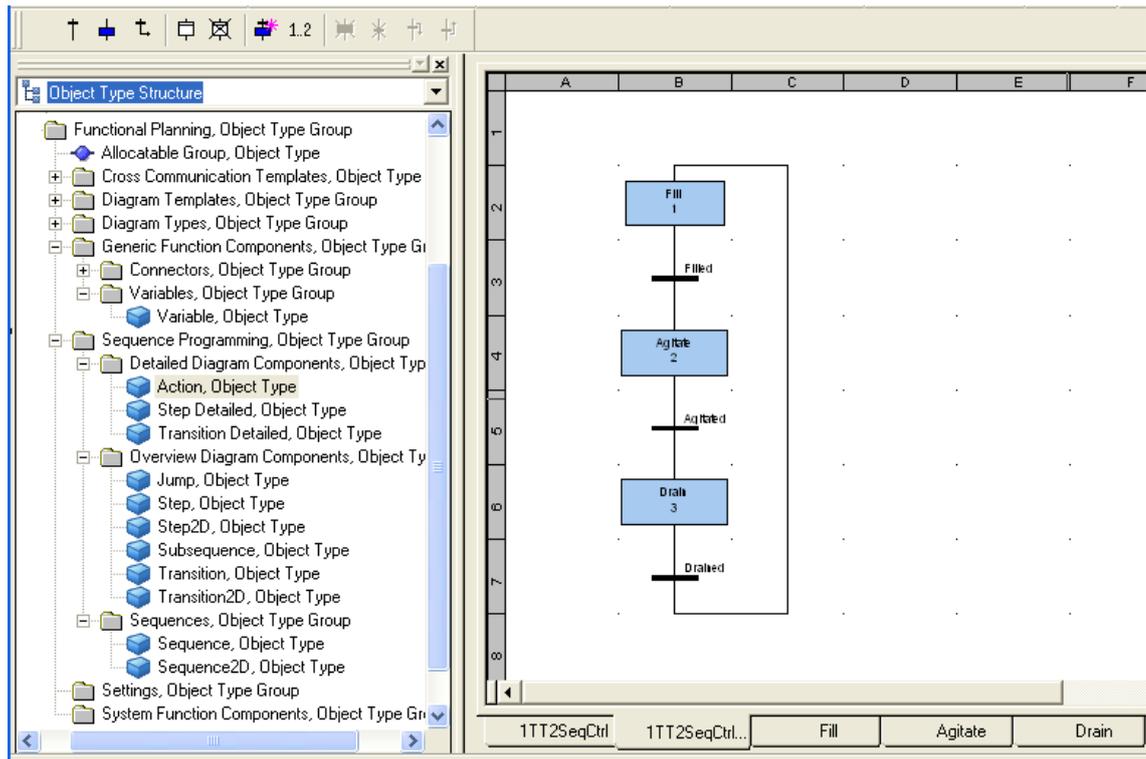


Figure 12. ABB 800xA sequential function chart (ABB, 2014, p. 66)

ABB 800xA is mostly compliant to the IEC 61131-3 standard. ABB 800xA supports parallel branches and alternative branches (ABB, 2014, p. 63). However, sequence transition logic must be programmed using FBD. ABB 800xA allows to save a function diagram as a template for later reuse (ABB, 2014, pp. 72-77). This feature also applies to SFCs.

2.3.4 CODESYS

The CODESYS sequence editor is a non-proprietary tool for configuring PLCs. It is developed and distributed by 3S-Smart Software Solutions GmbH (3S-Smart Software Solutions GmbH, 2019). CODESYS can be freely downloaded after registering to CODESYS webstore. The observations in this chapter are done by reading the manual and exploring the configuration software.

CODESYS editor implementation is the closest to the standard out of the tools evaluated in this thesis. It allows configuring the step actions in all the languages of the standard. The rising and falling edge pulse actions can be configured by selecting either “add entry action” or “add exit action” when adding the actions. CODESYS editor also has complete support for structured text (3S-Smart Software Solutions GmbH, 2019). Figure 13 displays the sequence configuration tool default view.

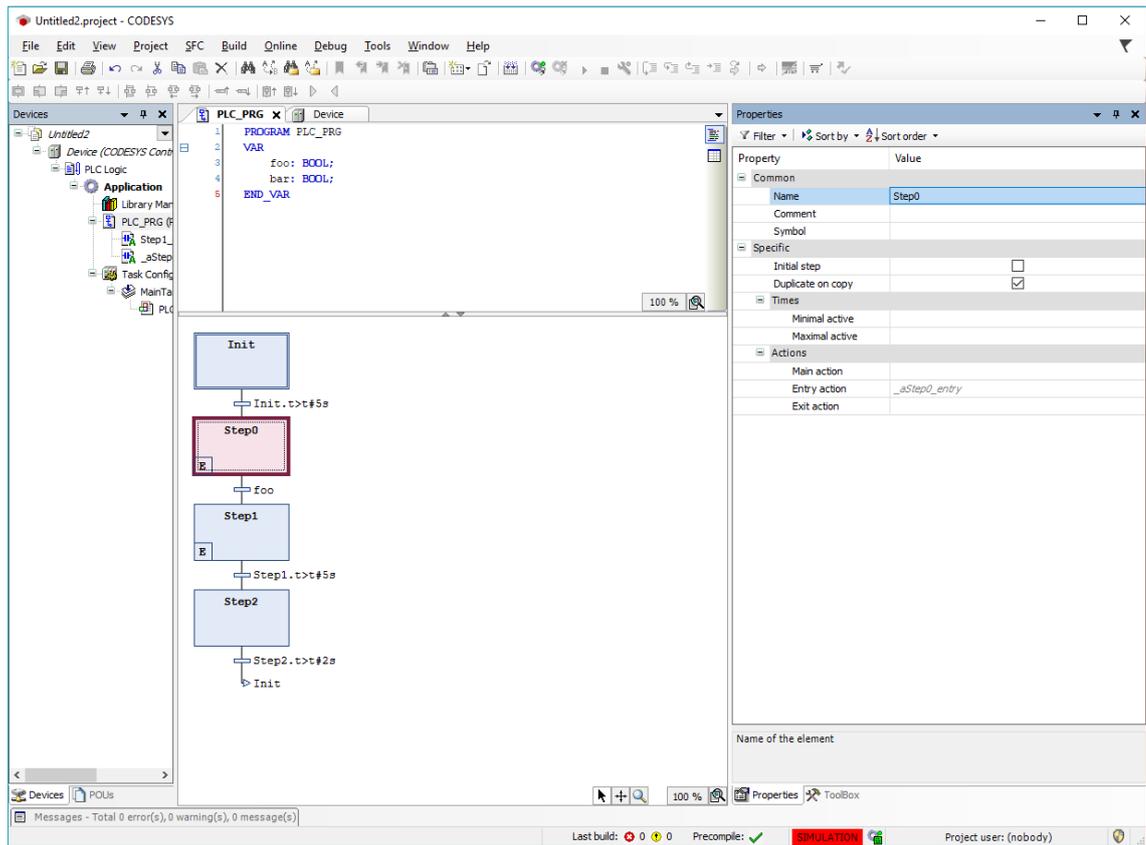


Figure 13 CODESYS sequence configurator

CODESYS configurator also includes a simulation mode, which enables easier debugging of the programs. In simulation mode the transition conditions are easy to verify to be correct. The software displays all the variables and the state of the sequence as can be seen in figure 14

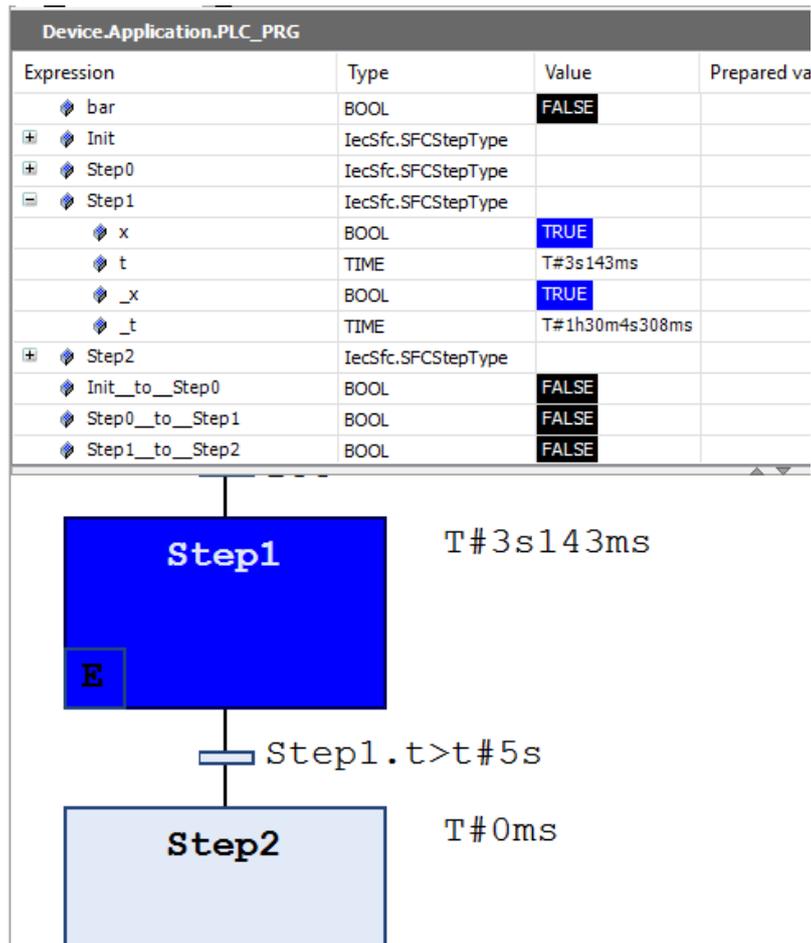


Figure 14 CODESYS SFC in simulation mode

Upper part of the figure displays runtime variable, their types and values. In the lower part the execution of the sequence can be monitored. In figure 14, Step1 is highlighted in blue, which means that it's currently active. In the figure, capital E in the bottom corner of the step represents that it has entry actions defined as a separate unit, in this case a ladder diagram. Actions could also be defined in the diagram in table format defined by the standard.

CODESYS is the most versatile of the programming tools discussed in this chapter. This is not surprising, given that it's non-proprietary tool. It must fit as many configuration workflows and schemes as possible for it to stay as widely used as it is today.

3 RESEARCH METHODS

The goal of the research is to find out how the IEC 61131-3 SFC standard could be improved to suit the needs of modern automation system. First step is to find out what is the needs for a SFC designer tool. This is best achieved by interviewing the users that are designing SFCs regularly.

The research will be qualitative by nature as the goal is to get in-depth analysis from experienced users about the flaws in the standard. Quantitative research methods are not efficient in this kind of research because the interest is in finding ideas and detailed opinions instead of quantitative measures. Additionally, it might prove to be difficult to find enough responders for quantitative methods to be efficient.

Exploring state-of-the-art research in the IEC 61131-3 standard shortcomings is vital to better understand the problem. The findings should be incorporated in to the interviews to better understand which of those improvements are something that the users require, and which ones are unimportant to the daily work of the engineers using the tools.

Benchmarking can also be used to help the interviews. When exploring different existing SFC designers, the emphasis is on the features that are not specified on the IEC 61131-3. Those features should be evaluated in the interviews to find out which are the must-haves and which can be dropped entirely.

After the interviews, a simple prototype could be built. This can be anything from as simple as paper prototype to something more refined like a simple demo software. Most probably, a hybrid of the two will be easiest to implement.

The prototype can then be used to verify the designs that emerged from the initial interviews. The prototype could be tested with a small audience, probably as a group interview as organizing two sets of interviews would require more time and more resources.

3.1 Qualitative research methods

The main tool in qualitative research is interviewing (Hirsjärvi, et al., 1997, p. 200). There's many different approaches into interviewing users. The approaches should be studied diligently to find the one that is best suited to this kind of research.

According to Hirsjärvi et al. (1997), interview is usually picked to be the research method if the research question is trying to map a largely unknown area and the answers might diverge in different directions. Interview is at its best when the answers are predicted to be complex and such that may need further clarification. (Hirsjärvi, et al., 1997, p. 200)

Hirsjärvi et al. (1997) state that interview has its characteristic downsides as a research method. The biggest one being that it's time consuming to plan and to execute. Interviews might also be a source of error in the research as all interviewees might not talk truthfully. In many cases the interviewee might give answers that presents them in the best light or overstate some issues if they suspect it has some positive effect on the study. (Hirsjärvi, et al., 1997, p. 201). These kinds of issues might not be as prominent in this research but might be good to take into consideration. The interviewers could be stuck in a rut with some way of doing things and thus might be conservative about the current features in the tools and not open to trying new methods.

Sandy Q. Qu and John Dumay (2011) state that two well established theoretical perspectives on research interview are neopositivism and romanticism. Neopositivist perspective focuses on studying the objective truth and sees the interview as a tool to collect data that is minimally biased (Qu & Dumay, 2011). Romanticist perspective on the other hand focuses on the interactivity of the interview and acknowledges that unbiased objective truth is next to impossible to find in the interviews (Qu & Dumay, 2011).

Interview can be either held as individual or focus group interviews. Focus group interviews are often less structured by nature and the interviewer serves as a moderator in the discussion. Focus group interviews are often easier to hold as they require less planning than most individual interviews. Ideally, the discussion flows freely, and interviewer only will have a couple of key questions in advance. Focus group interviews also save time as there's no need for as many sessions. (Qu & Dumay, 2011; Hirsjärvi, et al., 1997, pp. 205-207)

Whereas focus group interviews are always unstructured, individual interviews can be divided into three categories by their degree of structuration: structured, semi-structured and unstructured. Semi-structured is undescriptive as a category, because anything that is not structured or unstructured interview, falls into this category.

Careful planning is required of each of the interview methods, as often the interviewees are busy and the generally don't want to waste time in interviews (Qu & Dumay, 2011). This rings especially true in the context of this research. The project engineers that are configuring sequences have deadlines to meet and the more time spent on interviews the less time they have for working. With careful preparations and research, some of the questions can be omitted if the answer can be found from other sources. This saves the precious interview time and allows the interview to focus on the research question at hand (Qu & Dumay, 2011).

3.1.1 Structured interview

Structured interview method is very rigid as the interviewer is not supposed to deviate from the script at all. Structured interview consists of a set of predefined questions that

are asked always in the same order. This makes gathering the findings very easy from the transcripts. The interviewees may even be asked to pick the answer from a list, which makes reading the results even easier. Using predefined answers will however bring the interview closer to a quantitative rather than a qualitative research. (Qu & Dumay, 2011)

The neopositivist perspective on interviews prefers structured interviews as there will be next to none bias introduced by the interviewer (Qu & Dumay, 2011). On the contrary, the romanticist point of view sees the structured interview as too rigid to establish any kind of connection between the interviewer and the subject.

3.1.2 Unstructured interview

The opposite of the rigid structured interview is very informal unstructured interview. The goal of the unstructured interview is to make the interviewee feel comfortable. Unstructured interview does not require to have any of the questions thought in advance and it allows possible follow-up questions that might rise to the interviewer's mind. The questions may be worded differently and the discussion may flow entirely different way in each interview session. (Qu & Dumay, 2011)

Usually in unstructured interviews the interview sessions take more time than in unstructured or semi-structured interviews. Unstructured interviews are usually done in multiple separate sessions. This, and the fact that conducting unstructured interview requires a lot of skills from the interviewer, makes the unstructured interview an unpractical method for most master's thesis works. (Hirsjärvi, et al., 1997, pp. 204-205)

From the romanticist point of view, unstructured interviews are preferable to strictly structured interviews as better understanding of context may be achieved. In unstructured interviews, the difference in power and authority between interviewer and interviewee could bias the results. (Qu & Dumay, 2011)

3.1.3 Semi-structured interview

In between these two interview methods, lies the semi-structured interviews. Semi-structured interview is the most common interview method (Qu & Dumay, 2011). It is regarded as being flexible and convenient. It's usually easier to conduct because it requires less intense planning than structured interview and it's less demanding to the interviewer as unstructured interview.

Just like the unstructured interview, the semi-structured interview requires a skilful interviewer. As per Hannabuss' article Research interviews (1996), there are couple of important skills an interviewer must possess. First being the ability to create rapport with the interviewee. This helps to make the interviewees feel at ease and make them talk more freely. Second is keeping the discussion flowing with probing questions and avoiding

simple yes/no-questions or unnecessary jargon. Thirdly, the interviewer should be able to pace the interview by interrupting the direction of the conversation. At the same time, the interviewer should know when not to interrupt and let the silence play out for the benefit. The interviewer should also keep in mind to be non-judgemental and to have patience. The interviewer should not offer opinions and should refrain from giving non-verbal cues like nodding or looking surprised. (Hannabuss, 1996)

Semi-structured interview can have some predefined questions planned, but it leaves the questions open for modification. Different interviewees might interpret the same questions in different ways. Semi-structured interview method allows the interviewer to rephrase the questions or ask them in some different way when necessary.

Characteristically, the given answers are probed in semi-structured interviews to get more detailed answers. The interviewer might ask the interviewee to clarify some aspects of the answer or ask other clarifying questions when necessary.

The questions are often divided into 10 types. Each of them has a distinct purpose in the interview and skilful interviewer should know how and when to incorporate each type of question in each situation. The question types are summarised on table 4.

Table 4. *Types of interview questions (Qu & Dumay, 2011)*

Question type	Purpose
Introducing questions	To break the ice and start the interview
Follow-up questions	To keep the interview in the current topic
Probing questions	To get the interviewee to elaborate on the answer
Specifying questions	To get into the specifics and to get more precision out of the answer
Direct questions	To get direct and unambiguous answers
Indirect questions	To give opportunity for the interviewee to raise topics they see important
Structuring questions	To move the interview from one topic to another
Silence	To allow pauses for the interviewee to have time to reflect the topic

Interpreting questions	To clarify an answer and to make sure the interviewer interpreted the given answer correctly
Throw away questions	To build rapport or to keep the interviewee time to calm down.

The interview is started with the introducing questions, and telling some information about the interview. The interviewee doesn't need to be informed about the whole extent of the study beforehand, but it's good to at least tell what the data is used for and how the interview is going to play out. In some researches, the interviewees answers might be greatly influenced by the nature of the study at hand. That's why a balance should be found in giving just the right amount of information about the study. (Qu & Dumay, 2011)

3.2 Exploring literacy

There are quite a few proposed enhancements to the IEC 61131-3 standard already. As the standard features many different programming languages, only a few of these propositions are directly related to the SFC standard. However, as the languages are greatly intertwined, a change in some other language - especially ST, could have implications on SFC as well.

Mário de Sousa introduces several improvement propositions to the standard in his article Proposed corrections to the IEC 61131-3 standard (2010). One of the problems he sees in the standard is the lack of namespaces and too narrow definition of reserved keywords. What de Sousa is proposing, is adding basic namespace support to the ST language. Namespace is a very common concept that is found in many programming languages. De Sousa also states that the standard contains a few ambiguous syntax constructs that should be cleared out to make the standard more concise and unambiguous. Most of the ambiguities de Sousa lists are related to global variables and thus, not directly related to SFC programming. (de Sousa, 2010)

Jetley et al. (2013) suggest a method for controlling configuration versions by comparing the difference between previous and current version of the SFC program in their research paper "An approach for comparison of IEC 61131-3 graphical programs". The problem is that it's hard to find out what has changed in the graphical program as traditional text based approaches such as git cannot be used. (Jetley, et al., 2013)

The solution proposed by Jetley et al. (2013) starts with transforming the graphical programs into some other format, such as XML. Then the differences are very easy to detect programmatically. The bigger issue is how to determine which of these differences affect the program logic. A major change in some part of the code might cause minor implicit

changes in the latter code and showing all the changes as a difference will cause unnecessary noise and makes reading the difference harder. Jetley et al. (2013) have a pseudo-code algorithm which determines whether the change detected in the XML files is significant or if it can be ignored. (Jetley, et al., 2013)

In his book control of batch processes Smith (2014) explains the difficulties of controlling and maintaining batch processes. There are many characteristics that distinguish batch automation from continuous automation. For example, the recipes that are needed for making the products are usually made by different people that program the sequence to the plant. (Smith, 2014, pp. 291-292)

The problem in batch automation is that the process changes every time the produced product changes (Ferrarini & Piroddi, 2002; Smith, 2014, p. 290). Batch processing is used in manufacturing plants that create a variety of different specialized products. A typical example is dairy where the same equipment can output yogurt or sour cream depending on the process being used. Using own separate sequences for each product will be hard to maintain since a change in equipment demands changes in potentially numerous recipes.

Smith (2014) goes on to separate different phases the recipe usually goes through from the product design to making the product. The recipe usually starts as a corporate recipe, which doesn't rule in the equipment restrictions each production plant might have for example in terms of batch sizes. From there, the recipe is specialized to each production plant as a site recipe. Finally, the recipe must be converted to some PLC program, which executes the batch manufacturing using the equipment at hand in the production plant. (Smith, 2014, pp. 295-296)

Some industries, such as pharmaceutical or food manufacturing sets additional difficulties for batch automation. Batch automation in food and pharmaceutical industries often requires some version control system. Each country has its own regulations regarding the manufacturing of food or pharmaceutical products. One of the best known of such regulations is managed by the Food and Drugs Administration from the USA. In their Code of Federal Regulations title 21 part 11, is a detailed description of the measures a manufacturer should do to keep track of everything that might have an impact on the product quality (Food And Drugs Administration, 2018).

3.3 Benchmarking SFC tools

The purpose of benchmarking in this thesis is to gain an insight of the features present in some of the existing SFC tools. It means that the benchmarking is focused on solely on the quality of the features in the tested software and not on their performance. John Moses (2007) states that measuring software quality objectively and consistently is difficult and suggests some actions that can be used to improve the quality measurements. As the focus

in this research is not on measuring the quality as such, getting objective measurements is not so important in this research.

Shahizan and Feng (2005) proposes a useful tactic for developing the best tool is to utilize such methods as expert review and usability testing on the competitor products. This will expose the shortcomings of other similar products. (Shahizan & Feng, 2005)

4 END-USER INTERVIEWS

For this research, I chose to use semi-structured interviews as the research method. Semi-structured interview seems to have the right mix of flexibility and ease of implementation. Semi structured interviews offer a certain level of flexibility so the interview can be easily adjusted if the chosen topics or questions seem to be needing adjustments.

The structured interview is too rigid for any new ideas to emerge in the interviews. The interview would have to be focused on the evaluation of some predefined features and proposals. Structured interview is not suitable for trying to get new ideas to crop up. Additionally, structured interview would require a lot of interviewees to be effective. However, as the interviewees are often very busy and the research is done in the summer holiday time, interviewees might be hard to find.

Focus group interviews could be very effective for finding new ideas and faults in the standard. The problem of focus group interview is the same as with structured interviews. Finding interviewees and finding a time slot that is suitable for all the interviewees might be hard and thus organizing an effective focus group interview could be difficult. A small-scale focus group interview could be useful in the end phase to test the emerged designs, should there be enough time for that.

Unstructured interviews could be the best method for new ideas to emerge. On the other hand, unstructured interviews could turn out to be fruitless since it requires an experienced interviewer for it to be effective. Therefore, unstructured interviews were not chosen as the research method. However, keeping in mind the benefits of unstructured interview, it's probably good to design the interview to be closer to unstructured than structured interview.

4.1 Interviewed users

The goal is to find around 6 interviewees. After that, the interviews might start to repeat themselves and nothing new is gained. As earlier mentioned, finding the interviewees might be difficult in the summertime so even 8 might be a stretch. The lower limit for acceptable number of interviews is 4. Any less than that, the individual opinions become too influencing and the research cannot be generalized.

Ideally the interviewees will have some experience with multiple SFC tools. That way they probably have some view of the different features a SFC tool could have. If they have only been using a SFC tool from one vendor, they might not be so open to new features. However, in the extent of this study, it will be really hard to find interviewees outside the Valmet organization.

Primary source of finding the interviewees is from the Valmet automation service department. They are the end users in the resulting SFC tool so their opinions are valuable to include in the design. These are also the users that has likely the most experience of designing sequences.

Interviewees should also be searched elsewhere to get as diverse views as possible. Valmet automation research and development department has an application expert centre (AEC) that contains experienced users that are helping in the development of new tools. They could also have valuable opinions and possibly some knowledge of the IEC 61131-3 SFC standard.

If there's enough time and enough other interviewees, someone less experienced of the SFC standard could also be interviewed. That could help to deduce how easy the standard is to comprehend and how quick it is to learn. Interviewing those users is not the priority so these interviews should only be done if there's enough time to conduct them.

4.2 Interview topics

As the interviewed users are most probably unfamiliar of all the details of the IEC-61131-3 SFC standard, each of the topics should be started with explanation of the feature in question. The explanation should be kept concise to avoid bloating the interview session and to allow more time for the answers.

The focus of the interview should be kept in the ease of configuration. It's important that the sequence configuration tool makes configuring sequences as simple as possible. That would allow the engineer to focus on the logic that the sequence implements instead of how can it be achieved with the configuration tool. That will also help in making the maintenance easier and faster.

As stated earlier in chapter 3.2, the key thing that is missing from the standard is the support for templating. Different templating methods should be one topic in the interviews. Templating features however are not the kind of features that the standard should dictate. It is closer to the implementation details that each implementer should address in some way.

Also, stated in the chapter 3.2 was a possibility to statically check SFC syntax. The interviews could be used to determine how the sequences usually are tested. That could help to come up with a way that the testing could be made as easy and fast as possible.

The actions in a step is something that greatly varies from tool to tool. The ways to define actions should be made one of the key topics in the interviews. Action qualifiers is most likely to introduce opinions in the interviewees as it's a very important part of the execution of the actions and the abbreviations might be quite hard to remember or understand.

It should be made clear to the interviewees that the goal is only to find features that should be added to the standard, especially when interviewing people with programming experience. Otherwise the interviewees might suppress some of the ideas just because they think or know the feature to be too hard to implement.

4.2.1 Steps

As we learned in chapter 2.1.6, one of the key components of SFC is steps and transitions. There's quite a lot of intricate details related to the steps and transitions and it's not possible to go through all of them. Only the ones that might evoke some improvement ideas should be discussed.

Personally, I find the action definitions of a step hard to comprehend. It should be one of the topics discussed regarding steps. Different ways of defining actions should be introduced and evaluated. After discussing the existing ways, new better ways could be queried from the interviewee.

Transitions are a topic that likely won't be as opinionated as the actions. Interviews could be used to find out which one of the transition configuration methods is the best. For example, the interviewees might have a preferred language for writing Boolean logic. Although using structured text is a separate topic in the interviews, the ability to use structured text in transition conditions should be evaluated.

The interviewees should also evaluate is it beneficial to show the transition logic in the diagram or is it better to hide it. Hiding the transition conditions would make the diagram less cluttered, but could make the execution harder to follow.

4.2.2 Branches

Parallel and conditional branching will be interviewed as its own topic. Assuming the interviewees are only familiar using the Valmet DNA system, they might not have a clear understanding on how the branching works. Therefore, the rules for branching should be introduced first. Then the needs for this kind of branching should be canvassed.

Additionally, the limitations of branching should be discussed. Specifically jumping from one branch to another. The standard allows certain types of jumping but it's usually quite hard to determine if a jump is allowed or not. As an example, figure 15 contains an example of safe jump out of an execution branch.

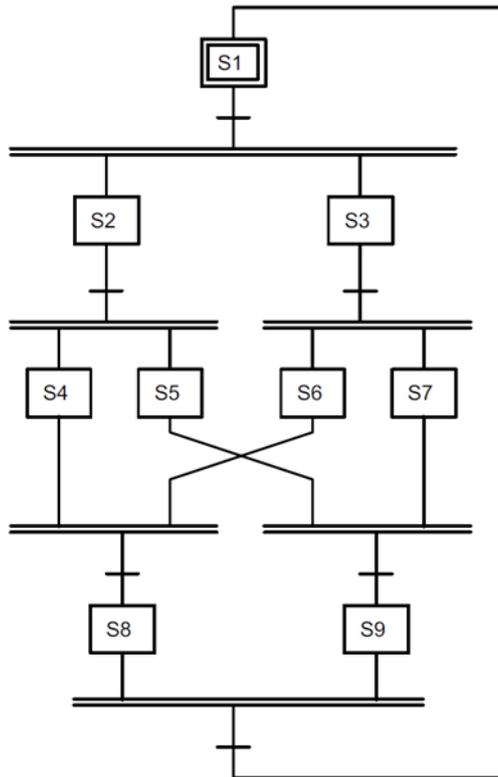


Figure 15. An example of safe wire network that has a jump out of branch (John & Tiegelkamp, 2010, p. 176)

The goal is to find out if there is any need for allowing jumping out of a branch. If there's no need for such a feature, it can be banned altogether and syntax checking will become much easier. The picture above should be used as an example in the interviews to illustrate what kind of jumping could be safe. Then the interviewees should estimate if that kind of jump would be necessary to be able to make.

4.2.3 Syntax verification

From the last topic of steps, illegal branch jumping, it's easy to transition into the syntax verification features. The standard does not include any syntax verification features. As discussed in chapter 3.2, there are some techniques that can be used to verify the syntax of SFC. These techniques should be evaluated from the interviewees after introducing them briefly.

Furthermore, the best way to show these errors to the user should be discussed. The most obvious ways of showing errors are either graphically by colouring the connections, steps or transitions in red or textually in some error descriptions. Textual and graphical representations are not necessarily exclusive as an editor can easily display both. Therefore, the goal is to find out which one the users find more useful.

The textual presentation of the error should also be discussed. It's not clear what the users want to see when there's an error. If the SFC can be converted back to ST, it might be useful to tell the line of code the error is in, but in graphical programming line numbers could be confusing.

4.2.4 Using structured text

Syntax verification discussion brings the conversation forward to using structured text for programming SFCs. One of the key features in IEC 61131-3 standard is that every programming language can be transpiled into structured text. This greatly improves flexibility but it can also create confusion in some users. The structured text could be introduced by first showing an example SFC and then showing a ST version of the same sequence. The example sequence should be kept simple and easy to comprehend as sequences defined in ST are usually quite simple.

Appendix A contains a simple example of sequence that is described in both SFC and ST languages. The sequence is a simplified version of starting a jet engine. The sequence contains only five steps and it doesn't have any branching so it should be simple enough to comprehend in limited time. The transitions conditions are also kept simple and they only contain simple number comparisons. Each step contains only one action and the actions are only simple setting and resetting a value into some input.

The interviewees should estimate, how big of sequence would be easier to configure with ST rather than graphically. The example above can serve in this purpose also. Looking at the example, the interviewee should be asked to estimate how many steps the structured text example should have to be easier to configure than the graphical one. The interviewees should also think if there's other factors that might favour using ST instead of SFC and vice versa.

It should be discussed in the interviews if the structured text language is found to be too complicated to understand. Also, the programming background of the users should be asked so the opinions of those users that are accustomed to textual programming could be separated from those who have next to none experience. If some users know only how to program using ST, they should be asked how easy the process of learning was.

4.2.5 Templating

Templates might be useful feature to have if similar kind of sequences tend to repeat. It can reduce the time it takes to configure a sequence. There are however some considerations that should be addressed, the most important being what should be templated. Is it the steps that are often like each other or is the whole sequence something that can be templated?

This part of the interview is not so much about verifying designs, but instead finding out what are the user needs and what are the sequences usually like. The goal is to find out what is the best way of using templates to make configuring sequences fast and simple.

Even with the help of templating, the steps cannot be entirely plug and play. Each system has a different kind of equipment and configuration and the steps need to have some knowledge about which inputs it needs to read and which outputs it needs to write to.

The interviewees might have some valuable information about just how much different systems have in common. That knowledge can be used to determine what kind of templates would be most helpful to engineers. A template with many parameters would probably not be any more efficient than programming the sequence by hand from the start.

4.2.6 Other

Managing recipes in batch automation usually produces a lot of redundant work and requires a lot of maintenance. The interviews can be used to find what are perceived to be the best tactics to maintain such recipes. The most prominent options are to use reusable templated step modules that can be reconfigured by changing some properties or to have a modular sequence which contain some sub-sequences that can be replaced when the recipe changes. Batch automation concepts and guidelines are defined in ISA-88 standard (ISA, 2019). That's why there's no need to dive further in to the subject in this research.

There's a significant chance that the interviewees are not experienced on batch automation needs as Valmet is not a big actor in the batch automation business. This can also be an advantage as the interviewers might not have any preconceptions on how the recipes should be configured in batch automation.

The newest addition to the IEC standard will improve transferability of control programs between different tools (PLCopen, 2013). The importance of this kind of interchangeability could be evaluated in the interviews.

4.3 Conclusion

The interview will go through many different topics. That's why it's important to timeslot each one of the topics carefully. One individual topic should not take too much time so there'll be enough time to go through each one of the topics. Table 5 contains the planned timetable for the interview topics. The timetable is not meant to be followed with exact precision, but it serves as a reference. The timetable is planned so that the interview takes about an hour. If possible, an hour and a half should be reserved to ensure the time doesn't run out.

Table 5. *Timetable for the interviews*

Topic	Target time in minutes	Total time in minutes
Actions	15	15
Transitions	15	30
Branching	5	35
Syntax verification	5	40
Structured text	10	50
Templating	5	55
Other topics	5	60

All the topics and supporting questions are collected in appendix B. The table on the appendix can be used to support the interview, but the interviews should not be limited only to the questions in the table. Instead the goal is to have the conversation flowing and trying to get the interviewee to come up with new ideas and new perspectives on configuring sequences.

5 RESULTS

The goal of this study was to find the weak and strong points of the IEC 61131-3 SFC standard. The standard was dissected into smaller individual parts to help the interviews. The most important parts are the steps and transitions, those were also the ones the interviewees had most opinions about.

As suspected earlier in the chapter 4, finding interviewees was hard, as much of the potential interviewees were quite busy with their own work. This study was therefore made with five interviews. Although small, the number of interviews was deemed to be sufficient as the same answers was starting to repeat as the interviews kept going. Therefore, it's sufficient to say that a saturation point was reached in the interviews.

The one-hour time slot that was reserved for each interview were appropriate and there was enough time to discuss each topic in every interview. Two of the interviews were a little shorter than anticipated and therefore did not produce as much data as the other interviews. This was most probably partly caused by the lack of experience from the interviewees part.

All the interviewees were from the Valmet organisation. The interviewees were searched from different departments to get opinions from different angles, therefore getting more diverse and throughout results. Two of the interviewees were from the services department, which mostly manage maintaining automation systems. Two of the interviewees were from the operations department, which do the initial configuration of the automation system from process description to control programs. Lastly, one of the interviewees was from the research and development department. All the interviewees were familiar with sequences and had been working with Valmet sequence CAD extensively.

As mentioned in the chapter 2.3.1, Valmet DNA sequence CAD does not support branching. Therefore, the discussions fell short on the branching topic since the interviewees were unexperienced with branching. For the same reason using structured text to configure sequences was not familiar among the interviewees. Fortunately, the example sequence that can be seen in the appendix A proved to be useful. It allowed the interviewees to evaluate how easy the syntax is to read.

In chapter 3.2, findings of other research on IEC 61131-3 SFC standard was introduced. These findings are discussed and evaluated further in later chapter 5.7. Just like the findings in this research, all the improvements or shortcomings are not something that can be fixed in the standard as they are closer to the implementation details. Rather they are something that each implementer should be aware of and consider when implementing a SFC configuration tool.

5.1 Steps

Step definitions were a topic that induced a lot of discussion in all the interviews. Specifically, the action definitions of a step were the topic that produced the most questions and opinions in this topic.

The interviewees found the function block diagram to be the preferable language of step action definitions. The action definitions in list form were also seen valuable particularly when there are a lot of simple actions in a step. A lot of the actions consist of writing Boolean values to some process inputs, this usually means starting or stopping motors or pumps. In such case filling a table was seen easier and quicker compared to adding and connecting block graphically. But then again, more complex actions were seen easier to configure using function block diagrams.

From user experience point of view, it could be confusing to the user if the actions could be configured in various ways. Therefore, it would make sense to limit the available languages to just one. However, if the benefits of table editing were wanted to be utilised, there could be an automatic conversion between table formatting and function block diagrams. Then the user could fill a table when the actions are simple and then the table is automatically converted to function block diagram. The user could also skip the table phase and construct a function block diagram from the scratch.

Structured text was not seen useful in defining an action. The reason was that structured text is too complicated to configure and too complicated to be read by the operator. Complex logic might be easier to write with structured text for an experienced engineer with programming skills, but the resulting program might be difficult to read by the operator and the sequence might be harder to debug in case of erroneous operation.

All the interviewees found the action qualifiers to be hard to understand. However, no better way of expressing them was found other than using the full name of the qualifier instead of just single letter. Mostly the interviewees could see a use case for each one of the qualifiers. The most important and useful qualifiers were seen to be set, reset and both pulse signals. There was however a concern raised by one interviewee that in pulse action, writing a value once to the system is not reliable enough as often messages sent through automation system can get lost. Therefore, it should either be possible to verify that the value got sent properly or sending it multiple times instead of just once.

Some of the interviewees proposed an addition to the action qualifiers, that there should be a qualifier that allows setting a non-Boolean value to an output. This would allow making simple writes while still using the simple table format. The written value could be expressed in the same manner as the delay time with delay qualifier. In the upper part of figure 16 there are actions defined using FBD language. Mostly the actions are just writing Boolean values to some inputs. The last action, however is writing a value 0,2 to

some output EIC-30:isp. The lower part of the figure displays the same actions defined using the table format. The last action in the table is the proposed syntax for writing non-Boolean values to an input. The syntax is like the one seen with delayed actions. The qualifier is represented with a *W*, and is followed by the value that should be set to the input.

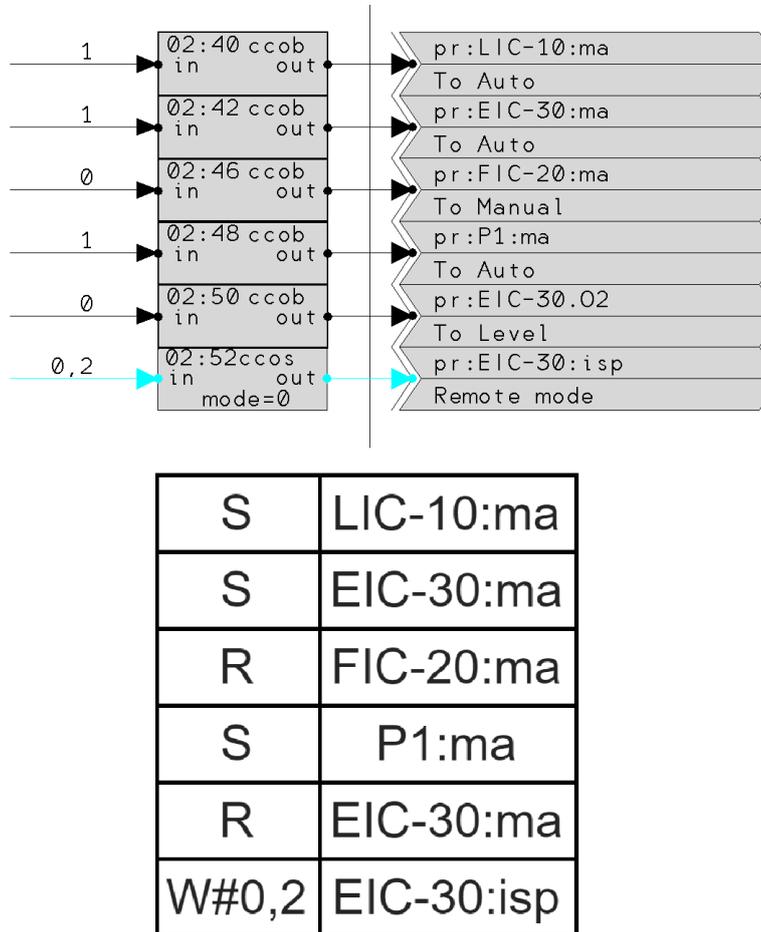


Figure 16. Actions defined as FBD and as a table

Defining an additional action qualifier would be beneficial, as it would broaden the actions that could be defined by simple table formatting. A single action qualifier for writing non-Boolean values would however fall short in cases where the value is not wanted to be set at the start of the step in the same way as with the qualifiers S and R. Expressing delays, pulses or time limited values would require either more action qualifiers or a way to chain them so that one action could have several qualifiers, given that they don't conflict with each other. As noted by the interviewees, the qualifiers are hard to understand as it is, so adding more would not be the optimal solution.

The interviewees found that it's better to have the action definitions visible in the sequence diagram. Then, it's easier to see what the sequence is doing in each step. It would make the development easier as there would be no need to jump between windows. It also makes the resulting diagram easier to debug.

In conclusion, the standard should be stricter about using different languages in step definitions so that a SFC would contain only one language instead of a whole spectrum of different IEC 61131-3 languages. This restriction would make the SFCs easier to read and maintain. Also, the action qualifiers should include a way to set non-Boolean values in the same ways as Boolean values in the S and R qualifiers. The non-Boolean set should be chainable to existing special qualifiers, such as limited time set, delay and pulse.

5.2 Transitions

FBD was also seen as the best language to configure the transition conditions of a step. The reason was that FBD is the easiest language to read both by the operators and the engineers configuring the system. Also ladder diagram was seen to be useful as it's as easy to read as FBDs.

Structured text was not seen as useful by the interviewees. The main reason was that the structured text expression might be hard to understand by operators that are trying to read the diagram. The transition conditions were seen to usually be too complex to write with a single structured text expression. Even though a single condition might be simple in a sequence, it would create confusion if one of the transfer conditions was written in ST and all others with FBD.

Mostly the interviewees thought that the transition conditions should be possible to hide from the diagram for example by using a connector and putting the transition condition in its own separate FBD. Showing the transition condition in the diagram would be beneficial, but could also cause a lot of clutter.

One feature missing from the standard that was seen essential for safety was the ability to set a maximum time that is waited for the condition to be fulfilled. This means an engineer could set some timeout after which an alarm is raised to the operator, who can act appropriately to ensure the process is set to a safe state. The alarm could also be accompanied with a jump to some predefined safety steps that set the process to a safe state. This further improves safety because it reduces the risk of human error and shortens the reaction time to errors.

This kind of feature is already possible to implement by the existing structures in the SFC standard by placing an additional branch from the step. The condition of this branch should check if the step has been on execution longer than the specified time. This can be

checked using the step variable T. An example in figure 17 displays the way to achieve the functionality by using the current SFC standard methods.

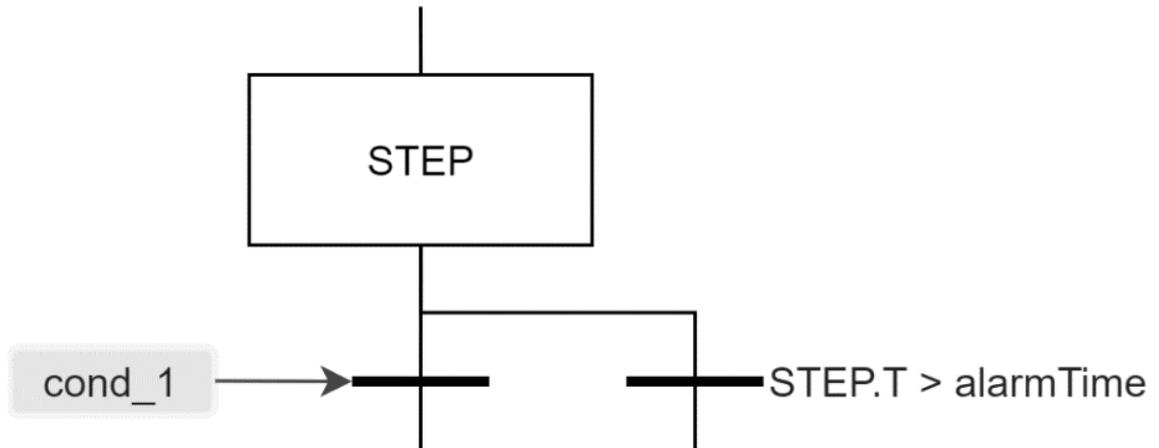


Figure 17. Jump to safety step after timeout has been reached, implemented in IEC 61131-3 compliant way

While the solution is seemingly simple, it could create a lot of clutter in to the diagram as the diagram gets bigger. Each step needs an additional branch and the conditions must be connected to some other part of the diagram, which further clutters the diagram. The other problem arises when there's a lot of other branches coming from the step. It could be difficult to distinguish the timeout branch from the other functional branches. Therefore, it would be better to either have some mechanism to configure error handling to the step itself or to make the error handling distinguishable from the other branches.

As with the action definitions of the step, it would be beneficial to define what is the main method of defining transition conditions. Based on the interviews, the main method should be function block diagrams. Other methods are not necessary to remove from the standard as they have some use cases in which they are better suited. The standard should however encourage the user to be consistent with the choice of language so that one diagram would only contain one language. This helps maintainability and readability of the SFC.

5.3 Branching

As previously mentioned, branching is not possible in the Valmet sequence CAD and therefore the interviewees didn't have much experience on the subject. Therefore, the interviewees were not able to give any in-depth answer on this topic. After explaining

how the branching works in the standard, the interviewees could estimate how useful the features might be.

All the interviewees did see a need for branching and thought that it was an important feature to be supported in a SFC tool. Conditional branching was seen more useful for parametrising the sequence and for reacting to different events from the process. Parallel branching was seen useful for grouping simultaneous actions together. Without parallel branches, all actions that needs to be executed simultaneously must be defined in a single steps actions. With parallel branching it's possible to group actions that are similar thus making maintenance and visualization easier.

As mentioned in chapter 2.1.6, there are some safe ways to jump out of parallel branch execution. However, none of the interviewees could see any practical use case for that kind of structure. Therefore, it would be easier to ban jumping out of parallel execution altogether. Only exception in this rule would be the error handling steps mentioned in the previous chapter. In case of the errors, jumping to the error step should stop the execution of all other parallel branches.

5.4 Testing

Verifying syntax in SFC can be achieved with either static syntax verification of simulating a process and running the sequence in that simulated environment. Most of the interviewees didn't see static checking nearly as important as test runs. The reason was that static syntax checking cannot find all the problems in the sequence. It can only really find problems in the SFC syntax such as missing parameter definitions or unconnected steps.

Test running was the main method to test the function of the SFC program. Test runs was seen to be more reliable method of testing and debugging SFCs. Manual testing is usually the only way of determining if the transition conditions are defined correctly. Some faulty conditions like *A AND not A* can easily be detected, but it's nearly impossible to see if some arbitrary combination of processing units can be on at the same time. Executing the sequence in a simulated environment requires the simulation environment to be constructed beforehand. Therefore, running simulations might not be worthwhile compared to testing with the real system in so called water runs.

Although testability of the control programs closely relates to the configuration, it wouldn't make sense to add these features to the standard. The standard is mostly related to only the configuration phase of the sequences or PLC programming in general. Therefore, it's closer to the implementation details of the sequence configuration tool rather than the standard.

5.5 Structured text

All except one of the interviewees were not familiar with structured text. Most of the interviewees were aware of the existence of structured text but had not been using it for any real work. The interviewees estimated that most automation engineers do not know how to program with ST. However, all the interviewees estimated that ST would be easy enough to learn given a good development environment with syntax suggestions and error checking.

Without exceptions, the interviewees saw the structured text notation of SFCs to be harder to understand than the graphical program. Graphical program is much easier to read and the visualisation of the sequence execution state is simpler.

The problem in using ST is that most the engineers that are using the tools are not familiar with it and thus would have to learn a completely different way of doing the same work. Also from the process operator point of view, the demand for an easy and expressive way to show sequences is even more important.

All the interviewees thought that configuring and editing sequences would be quicker with structured text. This however only applies to simple sequences with no branching, because a sequence with lot of steps and branches would need many transition conditions and following the execution logic would be hard.

The consensus was that the sequence configuration tool should be able to convert from structured text to graphical SFC easily and without additional configuration. That way the user would be able to get the best of both worlds, getting the swiftness of textual configuration and expressiveness of graphical languages.

This kind of feature is already possible with IEC 61131-3 as SFC can be expressed in ST. Doing the conversion to other direction from SFC to ST would require only a subset of the ST syntax to be available to the user, as discussed in chapter 2.1.6. A serious difficulty of this kind of implementation is determining the positions of each symbol in the graphical program. The tool could have automatic positioning for each element of the diagram, which would make it easier to convert from textual to graphical presentation. Figure 18 contains an example of what a sequence configuration tool might look like with automatic conversion from ST to SFC.

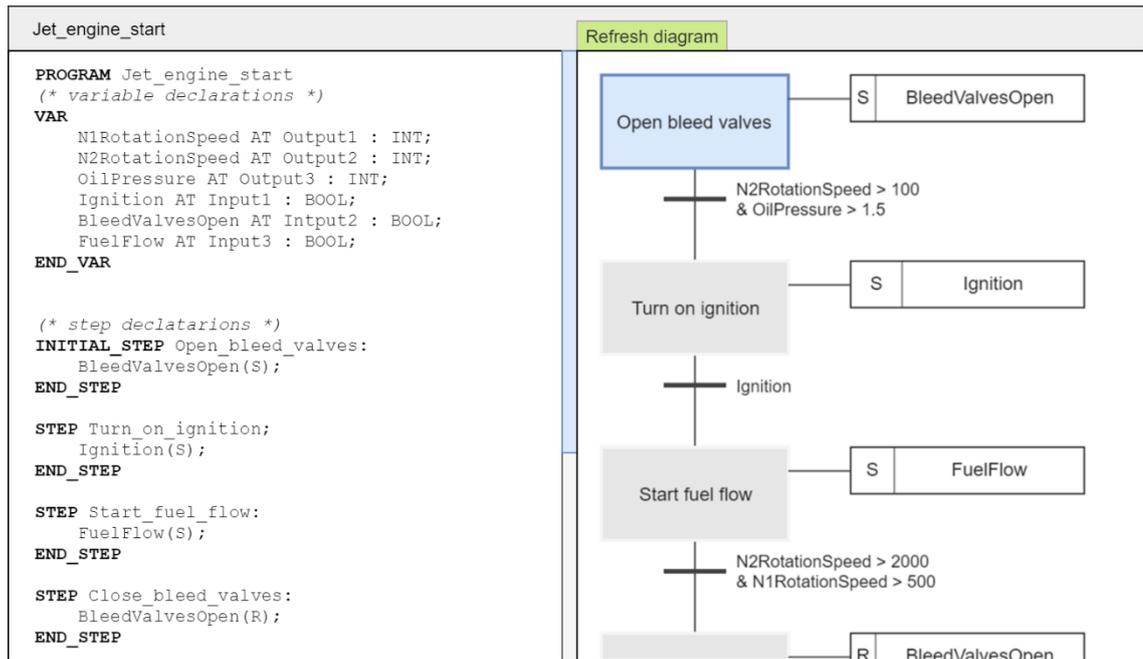


Figure 18. Sequence configuration tool view with ST to SFC compilation

In the example above, left side contains the sequence described in textual form and the right side has the same sequence expressed graphically. This example would be just one way to configure the sequence and the view could be opened from the editor separately so the users could choose what method they want to use. In this suggestion, the user first types the sequence or makes changes to existing sequence in the left panel. When ready, the user would press the refresh diagram -button and the diagram will be updated to match the textual presentation.

5.6 Templates

Templating the sequences and steps was a topic that turned in very different kind of results from different interviewees. Some of the interviewees found that templating would be beneficial and some did not find templating to be useful at all. Therefore, the results from this topic cannot be seen conclusive.

Some interviewees found that templates would make their job easier as the same base sequence could be reused with different parameters. They felt that the sequences are often very similar or that same kind of steps repeat in different sequences. In which case templating would increase code reuse and thus decrease redundancy.

However, some of the interviewees felt that the sequences are too hard to parameterize correctly to make templating possible. Each automation system has different kind of equipment and the equipment is often not in the same I/O position in different systems.

That's why each template would need at least as much parameters as there's inputs and outputs in the steps or the sequence. They also felt that the sequences needed in different automation systems differ too much from each other to make templates useful in their work.

5.7 Results from other sources

The improvements proposed by Mário de Sousa have miniscule effect on practical SFC configuration, based on the findings about STs lack of importance in configuring sequences. De Sousa was proposing adding namespaces to ST standard and clear some ambiguities regarding global variable usage.

The improvements proposed by Jetley et al. to have version control system in the standard was briefly discussed with the interviewees. They felt that there is no real need for comparing versions in SFC. Knowing which program the current iteration is, is all that the operators and engineers are going to need. They hardly ever try to figure out what is the difference between different iterations of the same programs.

The challenges introduced by the FDA regulations however suggests that version control is a very important feature to have in the configuration tools. Especially in food and pharmaceutical industries, it's important to keep track of all the changes that were made in each batch of the product. Version control, among other measures, allows the manufacturer to determine the cause, in case there's a faulty product dispatched to customers.

Although batch automation is closely related to SFCs, there are other standards such as ISA (international society of automation) S88 that addresses the issues and best practices when designing and maintaining batch process automation. Therefore, it wouldn't make sense to add any batch automation features in to the IEC 61131-3 standard other than version control. However, those standards and their requirements should be studied closely when implementing a SFC tool as sequences are a big part of controlling batch automation process.

5.8 Discussion

Although all the findings in this study are important to take into consideration when implementing a SFC tool, some of the findings are not something that should be noted in the standard.

All the interviewees were concerned of how the finished control program would look like to an operator. The standard doesn't really take a stand on how the program should be presented to the process operator. It is however a very serious concern that should be considered as the configuration of the system is only the first part in the lifecycle of the automation system. If the resulting program is not easy to read, there is additional work

that must be done to make an interface from which the status of the sequence is easy to read. The interviewees found that the graphical languages of the standard are self-documenting and that makes configuration faster as there's no need for additional efforts to make the program into readable form.

There would be some value in adding a mention of what should be the main language that is a must have in the implementation of the standard. This would make adopting the standard easier as it would be easier to choose what are the languages that the tool would have to support. Also, the engineers wouldn't need to learn all the different languages to be able to use various SFC tools.

As suspected earlier, the interviewees were quite stuck to their habits. Based on the answers the best way of configuring sequences is quite close to the way that they are currently doing it. While it's likely that these results apply to greater audience, it's good to consider that these interviewees were only used to working with the DNA sequence CAD. To draw more general conclusions about the standard itself, a more exhaustive study would be required. The interviewees should represent a more diverse group of people working with different implementations of the standard. Also, to get more exhaustive results the interviewees would have to be more educated on the standard either by studying the standard beforehand or by extending the interviews to give more information about the standard.

Comparing these results to the existing features in ABB, Siemens and Valmet sequence tools, will give a clue on how accurate these results are and what are the languages and working methods the automation engineers are used to working with. As all the tools explored in chapter 2.3 were missing the support for writing the entire sequence with structured text, it is safe to say that this feature is not seen as important by the implementers of the IEC 61131-3 standard.

More improvements could be uncovered, if more throughout research was conducted. The results from this study is however satisfactory, given the limited timespan and extent of this study. To cement the findings and possibly find new issues, the interviews should be conducted again, using a prototype that includes all the things found in the first interviews. Then the results could be verified to be correct and it's possible even to discover more results. The results are concluded in Table 6, where the subject column represents a finding from the interviews or other sources. Actions column represents what kind of finding it is.

Table 6. Conclusion of the results

Subject	Type
Non-Boolean writes in the actions	Addition to the standard
Alarm time limits to steps	Addition to the standard
Automatic conversion from ST to SFC	Implementation consideration
Actions shown in the diagram	Implementation consideration
Transition condition details hidden from the diagram	Implementation consideration

These results will be used in the development of Valmet DNA sequence editor. Some of the findings are missing features of the current implementation and some are just small refinements of current features. One thing clearly missing from the Valmet DNA sequence editor is conditional and parallel branching. These results can be used to make specifications to new features and improve the current features.

The newest addition to the IEC 61131 standard will be IEC 61131-10 that will be added at some point in 2019. The addition will introduce a xml format for saving IEC 61131-3 program units. A common format will help transferability between different sequence configuration tools. The format was formerly promoted by PLCOpen and the new standard is directly composed from that. (PLCOpen, 2013)

Adopting the standard might take a long time as automation providers use different tools which most probably use very different formats for saving and handling IEC 61131-3 programs. Adoption of new format might need major changes to the implementations of the tools. Also, some tools might have features that are not present in others.

6 CONCLUSIONS

Sequential controls are an important part of automation system. Lots of automation systems has actions or procedures that are supposed to be executed sequentially. Batch automation specifically has a need for sequential controls, but also continuous systems can greatly benefit from sequential controls. Sequences can be utilized when transitioning the system from one state to another. Sequences can also greatly improve the safety in the process as it will enable automatic safety shutdown procedures.

The IEC 61131-3 standard is widely used across different automation system providers. The standard defines the programming languages that can be used in programming PLCs. The standard includes a language that is used to program sequential control, sequential function chart. This language was the part of the standard that was under inspection in this thesis.

Usually the engineering workflow from investment to finished automation system follows the same patterns. First a process engineer designs how the process should work and makes descriptions of all the control applications that are needed to implement the automation system. From those descriptions an automation engineer will implement the control applications defined by the process descriptions. This is the part where all the IEC 61131-3 languages are needed. Then the control application is tested extensively in the real environment with “water runs” before taken into real use.

In this thesis the goal was to find shortcomings of the SFC language defined by the standard. The additional goal of this thesis was to find information that should be considered when implementing a sequence configuration tool. These are the kind of things that cannot be added to the standard or are not directly related to it, but still are important considerations in the implementation. The results will be used to aid the development of Valmet DNA sequence CAD.

The research was made by interviewing users that have previous experience using sequences. The interviews used were semi-structured, because it allowed extracting new information from the interviewees instead of just verifying currently available information. All the interviewees were not familiar with the standard and using semi-structured interviews was helpful as the standard could be gone through one topic at the time.

Most of the findings in the interviews were things, that should not be added to the standard either because they were not general enough or they were too close to the implementation details. However, there were lot of small notes that give an understanding on how to achieve better user experience in configuring the sequence.

The things that should be added to the standard include a way to define actions that can set non-Boolean values to variables or inputs. The way to achieve that, could be to add one additional action qualifier, that can be used with non-Boolean values and enabling a way to chain action qualifiers together. Without chaining possibilities, there would have to be many additional qualifiers just to enable non-Boolean values.

Other thing that should be added to the standard is the ability to add special handling for error situations. Usually it means the sequence is not able to pass to next step in the time it normally would. Error situations could also be more serious, such as equipment malfunction. In both cases, a special safety shutdown sequence should be executed to bring the equipment to a safe state. This kind of functionality is possible to achieve using current standard methods, but it includes lots of manual work and will introduce unnecessary complexity to the sequence. Having a special handling for exceptions would keep the sequence simple and therefore reduce programming errors and make operating the system easier.

The findings from the interviews also suggests that jumping out of a branch should not be possible. Interviewees found no use case for the example in Figure 15 where jumping out of a branch would not produce unsafe execution network. Only exception for this rule would be a shutdown sequence in case there's an error in the execution.

Important things to consider when implementing the functionalities of the standard include the relationship between structured text and graphical SFC program. The textual presentation of the program is not clear enough to be presented to the operators of the system, therefore it should not be used as the only way of configuring sequences. There are benefits in having ST as an option for programming the sequence, but it should be transpilable to graphical SFC.

Another thing that the implementer must address is what to show in the diagram and what to hide. The findings in this research suggests that the actions would be beneficial to be shown in the diagram to make the diagram more expressive. However, transition conditions should be omitted form the diagram to reduce clutter as the conditions might be often harder to fit in limited space.

All these additions and clarifications aim to help to make the implementation of the standard easier to use from the user's point of view. By considering the notions that was found in this research, configuring sequences will be fast and easy. Using the sequences would also be easy, since the operator's point of view is also considered.

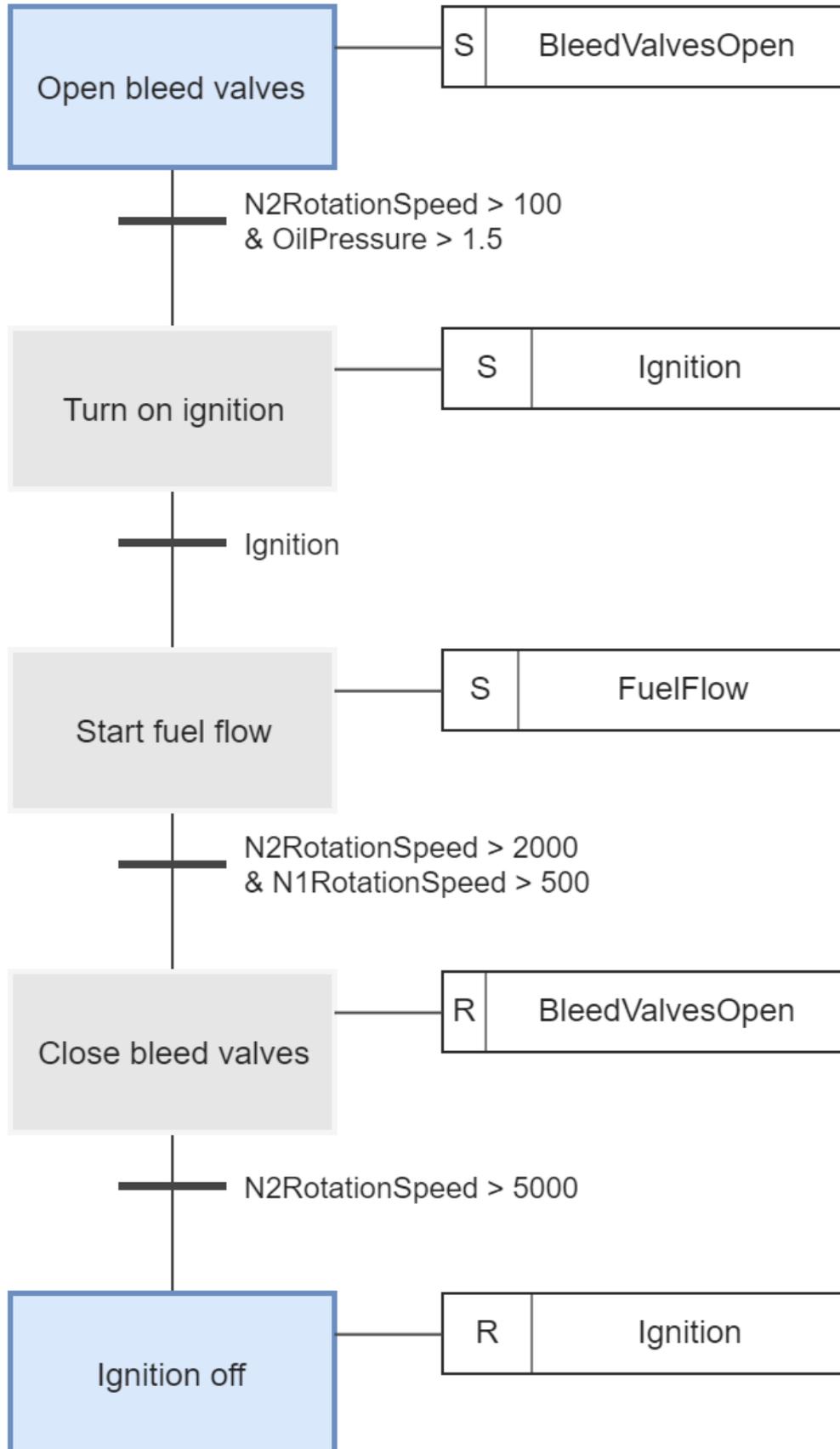
7 REFERENCES

- 3S-Smart Software Solutions GmbH, 2019. *CODESYS online help*. [Online]
Available at: <https://help.codesys.com/> [Accessed 16 February 2019].
- ABB, 2014. *System 800xA Configuration*. s.l.:ABB.
- Arnowitz, J., Arent, M. & Berger, N., 2010. *Effective Prototyping for Software Makers*. 1st ed. s.l.:Elsevier Science & Technology.
- Baker, A., Johnson, T., Kerpelman, D. & Sutherland, H., 1987. *GRAFNET and SFC as Factory Automation Standards Advantages and Limitations*. s.l.:IEEE.
- Bolton, W., 2009. *Programmable Logic Controllers*. s.l.:Elsevier science & technology.
- de Sousa, M., 2010. Proposed corrections to the IEC 61131-3 standard. *Computer Standards & Interfaces*, 32(5), pp. 312-320.
- Emerson, 2017. *Control Studio On-Line Product data sheet*. [Online]
Available at: <https://www.emerson.com/documents/automation/control-studio-online-en-57670.pdf>
[Accessed 2 February 2019].
- Ferrarini, L. & Piroddi, L., 2002. *Modular design and implementation of a logic control system for a batch process*. Milano, Italy: Elsevier.
- Food And Drugs Administration, 2018. *Code of Federal Regulations Title 21*. [Online]
Available at:
<https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfCFR/CFRSearch.cfm?CFRPart=11&showFR=1>
[Accessed 29 October 2018].
- Fuchs, J., Feldmann, S., Vogel-Heuser, B. & Legat, C., 2014. *Identification of Design Patterns for IEC 61131-3 in Machine and Plant Manufacturing*. Cape Town South Africa: IFAC.
- Fujino, K., Imafuku, K., Yamashita, Y. & Nishitani, H., 2000. *Design and verification of the SFC program for sequential control*. Nara, Japan: Elsevier Science Ltd.
- Hannabuss, S., 1996. Research interviews. *New library world*, 97(5), pp. 22-30.
- Hirsjärvi, S., Remes, P. & Sajavaara, P., 1997. *Tutki ja kirjoita*. 13th ed. Keuruu, Finland: Tekijät ja kirjayhtymä Oy.

- Honeywell, 2018. *ControlEdge PLC Specification*. [Online] Available at: https://www.honeywellprocess.com/library/marketing/tech-specs/CE03-100-150.1_V1.3.pdf [Accessed 2 February 2019].
- Hrúz, B. & Zhou, M., 2007. *Modeling and Control of Discrete-event Dynamic Systems with Petri Nets and Other Tool*. s.l.:Springer-verlag London Ltd..
- Huffman, D. A., 2015. *The Role of Sequential Automation in Improving Process Safety*. Wickliffe, Ohio, USA: Process Automation Division, ABB Inc.,.
- IEC, 2013. *IEC 61131-3: Programmable controllers - Part 3: Programming languages*. 3rd ed. Geneva, Switzerland: IEC.
- IEC, 2018. *About the IEC*. [Online] Available at: <http://www.iec.ch/about/> [Accessed 23 May 2018].
- ISA, 2019. *ISA88, Batch Control*. [Online] Available at: <https://www.isa.org/templates/two-column.aspx?pageid=124544> [Accessed 16 February 2019].
- Jetley, R. et al., 2013. *An Approach for Comparison of IEC 61131-3 Graphical Programs*, s.l.: IEEE.
- John, K.-H. & Tiegelkamp, M., 2010. *IEC 61131-3: Programming Industrial Automation Systems*. Berlin: Springer.
- Johnsson, C. & Årzén, K.-E., 1999. Grafchart and grafcet: a comparison between two graphical languages aimed for sequential control applications. *IFAC Proceedings Volumes*, 32(2), pp. 19-24.
- Lewis, R. W., 1998. *Programming industrial control systems using IEC 1131-1 Revised edition*. Exceter, England: Short run press Ltd..
- Lydon, B., 2012. IEC 61131-3 industrial control programming standard advancements. *Intech*, Issue September-October, pp. 30-33.
- Moses, J., 2007. Benchmarking quality measurement. *Software Quality Journal*, 15(4), pp. 449-462.
- Okuda, M., Nagao, T., Mizuya, T. & Miyazawa, I., 2013. *Research on Structural Analysis for the Petri-net representing Sequential Function Chart*. Kanagawa, Japan: SICE.

- PLCopen, 2013. *PLCopen*. [Online] Available at: http://www.plcopen.org/pages/tc1_standards/downloads/ [Accessed 8 May 2018].
- Pratt, G. L. & Tripplett, T. L., 2016. OPC UA and IEC 61131-3. *Intech*, Issue September-October, pp. 30-33.
- Qu, S. Q. & Dumay, J., 2011. The qualitative research interview. *Qualitative Research in Accounting & Management*, 8(3), pp. 238-264.
- Schumacher, F. & Fay, A., 2011. *Requirements and obstacles for the transformation of GRAFCET specifications into IEC 61131-3 PLC programs*. Toulouse, France, IEEE.
- Shahizan, H. & Feng, L., 2005. Evaluating the Usability and Content Usefulness of Web Sites: A Benchmarking Approach. *Journal of Electronic Commerce in Organizations*, 3(April-June), pp. 46-67.
- Siemens AG, 2017. *SIMATIC STEP 7 and WinCC Engineering V15 System Manual*. Nürnberg, Germany: Siemens AG.
- Smith, C. L., 2014. *Control of batch processes*. 1st ed. s.l.:John Wiley & Sons, Inc.
- Valmet Automation Oy, 2017. *Valmet DNA Engineering: Sequence CAD -käyttöohje*. Tampere, Finland: Valmet Automation Oy.
- Öhman, M., Johansson, S. & Årzén, K.-E., 1998. *Implementation aspects of the PLC standard IEC 1131-3*, Lund, Sweden: Department of Automatic Control, Lund Institute of Technology.

APPENDIX A: EXAMPLE SFC WRITTEN IN ST



```

PROGRAM Jet_engine_start
  (* variable declarations *)
  VAR
    N1RotationSpeed AT Output1 : INT;
    N2RotationSpeed AT Output2 : INT;
    OilPressure AT Output3 : INT;
    Ignition AT Input1 : BOOL;
    BleedValvesOpen AT Input2 : BOOL;
    FuelFlow AT Input3 : BOOL;
  END_VAR

  (* step declatarions *)
  INITIAL_STEP Open_bleed_valves:
    BleedValvesOpen(S);
  END_STEP

  STEP Turn_on_ignition;
    Ignition(S);
  END_STEP

  STEP Start_fuel_flow:
    FuelFlow(S);
  END_STEP

  STEP Close_bleed_valves:
    BleedValvesOpen(R);
  END_STEP

  STEP Ignition_off:
    Ignition(R);
  END_STEP

  (* transition declarations *)
  TRANSITION FROM Open_bleed_valves TO Turn_on_ignition :=
    N2RotationSpeed > 100 & OilPressure > 1.5;
  END_TRANSITION

  TRANSITION FROM Turn_on_ignition TO Start_fuel_flow :=
    Ignition;
  END_TRANSITION

  TRANSITION FROM Start_fuel_flow TO Close_bleed_valves :=
    N2RotationSpeed > 2000 & N1RotationSpeed > 500;
  END_TRANSITION

  TRANSITION FROM Close_bleed_valves TO Ignition_off :=
    N2RotationSpeed > 5000;
  END_TRANSITION

END_PROGRAM

```

APPENDIX B: INTERVIEW TOPICS AND SUPPORTING QUESTIONS

Topic	Questions
<p>Actions</p> <p>15 min</p>	<p>Introduction</p> <p>In which language, you would prefer to define the actions?</p> <p>Is there some language you don't see valuable in action definitions?</p> <p>Should the actions be shown in the chart?</p> <p>Do the actions make the chart easier to read or create clutter?</p> <p>Are the action qualifiers easy to comprehend?</p> <p>Can you see any better ways of expressing them?</p>
<p>Transitions</p> <p>30 min</p>	<p>Introduction</p> <p>What is the best language to define the transitions?</p> <p>What do you see as the primary language for describing transitions?</p> <p>Is there any language that is not necessary?</p> <p>Are there any benefits for having the transitions outside the step instead in the step description like with actions?</p>
<p>Branching</p> <p>35 min</p>	<p>How often are branches needed?</p> <p>Is there a need to jump out of branch?</p>
<p>Syntax verification and version diffs</p> <p>40 min</p>	<p>Introduce basic methods</p> <p>Is showing textual error messages enough or should there be some indicator in the diagram?</p> <p>What the error message should contain?</p> <p>How the differences should be displayed?</p>

<p>Structured text</p> <p>50 min</p>	<p>Is structured text familiar?</p> <p>What are the programming languages that you have used?</p> <p>Introduce structured text: the base of every other language by showing the example in appendix A.</p> <p>Discussion about which one is better</p> <p>What are the pros and cons in each one?</p> <p>How big of a SFC would be easier to write using ST?</p> <p>Is there some value in ST representation in debugging etc.?</p> <p>Is structured text easy enough to understand?</p>
<p>Templating</p> <p>55 min</p>	<p>How similar the sequences are to each other?</p> <p>How much the steps have in common?</p> <p>Would templates speed up the work?</p> <p>Should the steps be as a template or the whole diagram?</p>