REQCAP: Hierarchical Requirements Modeling and Test Generation for Industrial Control Systems

Ali Almohammad*, João F. Ferreira†‡, Alexandra Mendes† and Phil White*
*Applied Integration UK Ltd, Stokesley, TS9 5JZ, UK
†School of Computing, Teesside University, Middlesbrough, TS1 3BX, UK
‡HASLab/INESC TEC, Universidade do Minho, 4704-553 Braga, Portugal

***Abstract*—This paper presents REQCAP, an implementation of a new method that articulates hierarchical requirements modeling and test generation to assist in the process of capturing requirements for PLC-based control systems. REQCAP is based on a semi-formal graphical model that supports hierarchical modeling, thus enabling compositional specifications. The tool supports automated generation of test cases according to different coverage criteria. It can also import requirements directly from REQIF files and automatically generate Sequential Function Charts (SFCs).**

We use a real-world case study to show how REQCAP can be used to model realistic system requirements. We show how the automated generation of SFCs and test cases can support engineers (and clients) in visualizing and reviewing requirements. Moreover, all the tests listed in the original test document of the case study are also generated automatically by REQCAP, demonstrating that the tool can be used to effectively capture requirements and generate valid and useful test cases.

I. INTRODUCTION

Programmable logic controllers (PLCs) are widely used in safety-critical processes, such as gas monitoring and ventilation control in mines [19], administration and treatment of municipal and industrial wastewater [10], and nuclear power plants [4], [15]. The reliability of PLC-based control systems is therefore extremely important. However, they can be very complex, making it difficult to precisely capture all their safety requirements and ensure their reliability.

A model commonly used in the development of PLC-based control systems is the V-model, an extension of the waterfall model where validation, verification, and testing are planned and performed in parallel with requirement gathering, design, and implementation phases (respectively). The V-model is simple, easy to use, and has the advantage of including some validation before coding takes place. However, if any errors are found during the testing phase, then the test documents along with requirement documents have to be updated. Depending on the complexity of the project, and given that in practice the modeling is often done manually, changes can be expensive. Moreover, a problem that arises in practice is that different engineers often use different notations and methods to capture requirements. This hinders reusability and compositionality, making it more difficult and expensive to develop large-scale projects.

This work has been supported by Innovate UK funding as part of the Knowledge Transfer Partnership no. 9828.

Model-based design, whereby a single model of the entire system is used throughout the development process, is an approach that can help alleviate some of the problems mentioned above. For example, it provides a common design environment and framework for communication across project teams and integrates testing with design, allowing continuous detection and correction of errors. The model is an executable specification which can be used to explore alternative designs and evaluate system performance, thus yielding a refined, optimized, and fully tested software.

In this paper, we present REQCAP, a model-based requirement modeling tool that implements a new method that can be used to effectively capture requirements of PLC-based systems. The method emphasizes the procedural nature of control systems, allowing a simple, precise, and uniform approach to capture systems' requirements. REQCAP supports automated generation of test cases according to different code coverage criteria. It can also import requirements directly from REQIF files [23] and it is capable of automatically generating Sequential Function Charts (SFCs). We also outline some challenges that we encounter in practice. We believe that by reporting our experience in developing this tool, we are providing valuable information to the community. This work was developed in an industrial context and captures best practice gathered by years of experience delivering real PLC-based projects. It puts together some features that facilitate the task of requirements engineers. Some of these features include:

- a uniform graphical semi-formal model to capture requirements, emphasizing procedural modeling and promoting model reusability;
- different levels of abstraction, supported by the hierarchical structure of the models;
- automatic test generation according to different coverage criteria;
- automatic generation of Sequential Function Charts (SFCs).

To the best of our knowledge, no other tool allows the combination of all these features.

We use a real-world case study to show how REQCAP can be used to model realistic system requirements. We discuss how the automated generation of SFCs and test cases can support engineers (and clients) in visualizing and reviewing requirements. Moreover, all the tests listed in the original test

document of the case study are also generated automatically by REQCAP, demonstrating that the tool can be used to effectively capture requirements and generate valid and useful test cases.

This paper is structured as follows. In Section II, we explain the context in which the tool was developed and outline some of the challenges involved in the development of PLC-based systems. We then describe the approach and tool developed in Section III. In Section IV, we show that REQCAP can be used to model realistic system requirements by using a real-world case study. After presenting and discussing some relevant related work in Section V, we conclude the paper and set some future directions in Section VI.

II. BACKGROUND AND CHALLENGES

The work described in this paper is developed in an industrial context (at Applied Integration UK) and captures best practice gathered by years of experience delivering projects that depend on programmable logic controllers (PLCs). Applied Integration UK are approved integrators for all the leading PLC manufacturers, delivering systems applicable across the oil, gas, petrochemical, and nuclear energy industries. The company provides guidance on how to implement PLC Safety Systems, the use of safety functions and compliance with industry standards, as well as the total integration of systems.

PLCs are computer control systems that have a cyclic execution model: they continuously monitor the state of input devices and make decisions based upon a custom program to control the state of output devices. There are four basic steps in the operation of all PLCs: (1) Human and sensor input is collected (Input Scan), (2) some user-created program logic is executed (Program Scan), (3) results are passed to the environment (Output Scan), and (4) some housekeeping is performed, such as communication with programming terminals and internal diagnostics (Housekeeping). These steps continually take place in a repeating loop.

PLCs can be programmed using different programming languages. The International Electrotechnical Commission (IEC) 61131-3 standard [5], [16] was introduced in the 1990s in an attempt to standardize programming languages for industrial automation. The standard recommends the specification of the syntax and semantics of a unified suite of programming languages, including the overall software model and a structuring language. The programming languages are: Ladder Diagram (LD), Function Block Diagram (FBD), Structured Text (ST), Instruction List (IL) and Sequential Function Chart (SFC).

In terms of development methodologies, PLC-based control systems are usually developed following the V-model. As explained in the previous section, the V-model is an extension of the waterfall model where validation, verification, and testing are planned and performed in parallel with requirement gathering, design, and implementation phases (respectively). The V-model is simple, easy to use, and fits well with the development of PLC-based control systems because it emphasizes validation. Indeed, an important task in the development of PLC-based control systems is the validation of the program logic, particularly in safety-critical systems. Test documents are normally written during the design phase. These tend to be long, detailed, and manually produced by (requirements) engineers. Before the implementation phase starts, the test documents are normally used (together with other design documents) to clarify requirements with clients. They are also used to inform the implementation phase. Testing of the implementation is often done at the end of the project lifecycle, using simulation techniques, and it is usually performed manually by a testing engineer.

The lack of automation in validation is a major factor in the cost of projects. Producing the test documents, where all test cases are listed, takes a long time. Moreover, it is often the case that standard code coverage criteria are not used by engineers to produce the set of test cases.

The points above show that there are some challenges involved in the design of PLC-based control systems. Based on the experience of engineers working at Applied Integration UK, on related research literature (e.g. [18], [25]), and given the context of this project, the following challenges were identified as the most relevant:

1) Lack of standard modeling and development approach: engineers often use different, improper, and/or mixed-granularity abstractions, leading to a lack of consistency, performance, and progress measures across projects;
2) High cost of verification and validation and lack of code-coverage consistency across projects;
3) Lack of automation, both in terms of generation of test cases, and in terms of generation of documentation and other important design artefacts (e.g. SFCs and flowcharts).

III. FROM REQUIREMENTS TO TESTING

In this section we describe some important aspects of REQCAP, focusing on the challenges outlined in the previous section. The development of this work was informed by a focus group that consists of seven engineers: two directors with a background on the development of complex control systems, three senior control engineers, one junior control engineer who has experience with software testing, and a software engineer. The project was also accompanied by an academic team from Teesside University who specialise in formal methods in software engineering.

A. Modeling

There are two important points that emerged from Applied Integration's experience in delivering PLC-based control system projects and that were discussed and confirmed in the focus group meetings. The first is that the modeling formalism should highlight the procedural nature of control systems, clearly capturing the dependencies between components. This idea is not new: there is related work on procedural modeling based on finite state machines, on Petri nets, and on procedural domain-specific languages [11], [9]. The second point is that it should be possible for engineers to view the model at different abstract levels. This idea follows existing practice at Applied

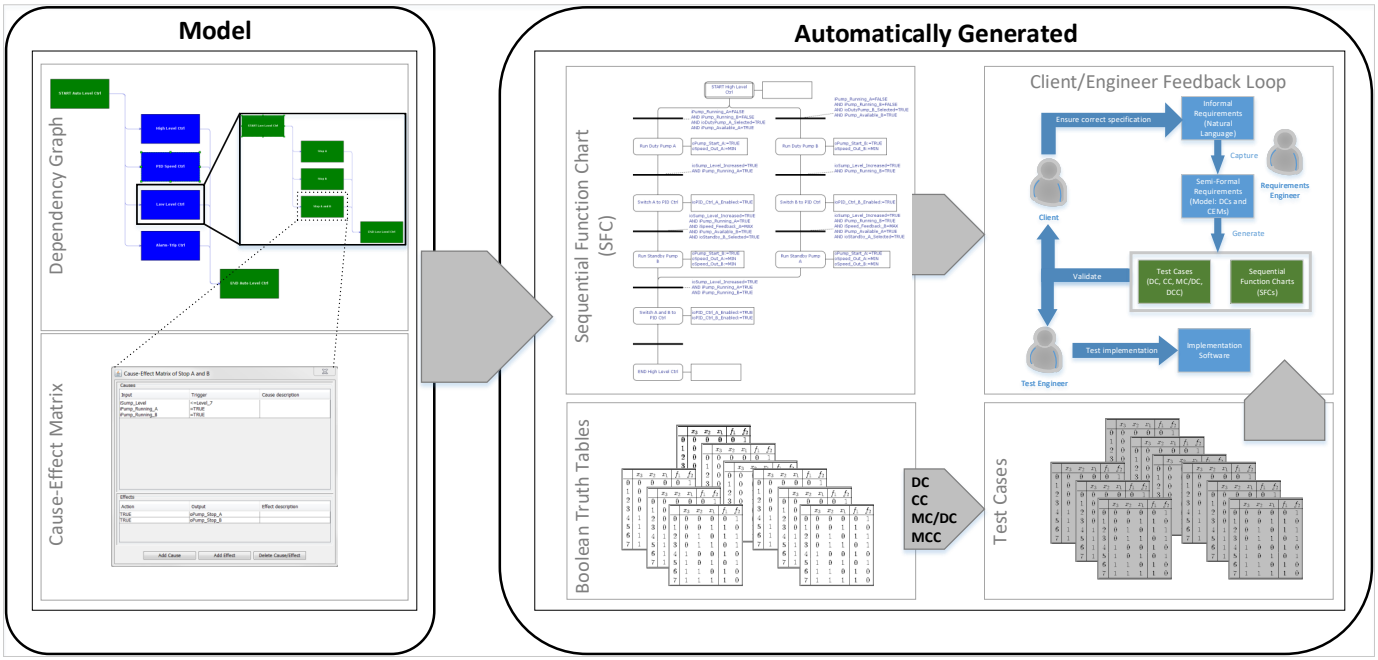


Fig. 1: In REQCAP, dependency charts (DCs) and cause-effect matrices (CEMs) are used to model systems. From these, sequential function charts (SFCs) and test cases satisfying different coverage criteria are generated. The generated artefacts can support engineers and clients visualizing and reviewing requirements.

Integration, but there is also literature supporting this (e.g. [17]).

REQCAP supports these points by providing a graph-based graphical language that captures dependencies between procedural control entities. These entities can be refined as required, allowing engineers to analyze the model at different abstract levels. We call such graphs *hierarchical dependency charts* (DCs). To formally specify these control entities, we use *cause-effect matrices* (CEMs). These are described as follows.

a) Hierarchical dependency charts (DCs): The *graphical model* consists of a root DC that describes the complete process from a high-level perspective. Nodes in the root DC usually represent high-level components that are further refined in more specific DCs. In general, DCs consist of the following elements:

- *Start* and *End* nodes: every DC represents a functional unit, so REQCAP requires that the start and end points are explicitly shown;
- *High-level* (blue) nodes: these nodes represent non-atomic functional units, meaning that they are further refined;
- *Low-level* (green) nodes: these nodes represent atomic functional units and can be formally defined using cause-effect matrices.

For example, Figure 1 depicts a DC with four high-level (blue) nodes. The refinement of the node *Low Level Ctrl* into a DC with only low-level (green) nodes is also shown. Both DCs have Start and End nodes.

b) Cause-effect matrices (CEMs): High-level (blue) nodes can be iteratively refined to a point where no further refinement is possible or desirable (i.e. to a point where a low-level (green) node is reached). At that point, the node can be formally defined. In REQCAP, this is done by defining a cause-effect matrix (CEM), which is a tabular representation used to specify logically the relations between causes (inputs' conditions) and effects (outputs' conditions). This formalism has been chosen because it is easy to understand and intuitive to use. Moreover, it is widely used in industry. The CEM form that we use in this work is a simplified form of the original *cause-effects graph*, a popular formalism documented in the literature [21], [22]. Our CEM form represents the logical relation between causes and effects using a restricted form that is very similar to disjunctive normal form (DNF) in Boolean logic. The main effect(s) can be seen as a sum of products of causes. For example:

Causes			Effects
A = True	AND	OR	Z:=True
x >-15			
y <15.5	AND		
B = False			

Fig. 2: Example of a cause-effect matrix

$$Z = (A \text{ and } (x>-15)) \text{ or } ((y<15.5) \text{ and not } B)$$

The tabular form of this example is depicted in Figure 2. Figure 1 shows how CEMs are displayed in REQCAP.

B. SFC Generation

Although DCs and the associated cause-effect matrices (CEMs) are created by domain experts (control engineers), the simplicity of their notations and constructs make it possible to communicate model requirements to the client at the early stages of the project — even though clients are not expected

to be familiar with these. This communication with the client allows requirements engineers to collect client feedback on the model and identify any missing elements. Well-established domain-specific languages, such as SFC [16], are expected to be more familiar to clients than DCs. SFC is a graphical language used for programming PLCs and, as mentioned in the previous section, it is one of the five languages defined by IEC 61131-3 standard [5]. SFCs can be used to describe the sequential behaviour of control systems at the top level in terms of states, all the possible state transitions, and transition conditions. In REQCAP, we have implemented an algorithm that generates an equivalent high-level SFC from DCs and CEMs.

C. Test Case Generation

Test coverage [13] is a measure used to describe the percentage of code (or model, in this context) that has been exercised/visited when test cases run. REQCAP implements four coverage criteria:

1) Decision Coverage (DC)
2) Condition Coverage (CC)
3) Modified Condition/Decision Coverage (MC/DC)
4) Multiple Condition Coverage (MCC)

These are extensively discussed in the literature, but to briefly explain the differences between them, consider the decision *(A or B)*, where *A* and *B* are both conditions. In general, decisions are made up from conditions combined by logical operators (such as *and*, *or*, *not*). DC requires two test cases: one for a true outcome and another for a false outcome of the decision. CC requires that each condition in a decision takes on all possible outcomes at least once, but does not require that the decision takes on all possible outcomes at least once. MC/DC requires that each condition is shown to independently affect the outcome of the decision. Therefore, each condition in the decision requires two test cases that: **(R1)** differ in the outcome for that condition, **(R2)** have the same outcomes for all other conditions, and **(R3)** produce *true* and *false* in the outcome of the whole decision. For n conditions, MC/DC requires between $n+1$ and $2n$ test cases. Finally, MCC requires that all combinations of true and false for the conditions in the decision are evaluated.

In REQCAP, the user can select specific test coverage criteria (by default, MC/DC is used). Test cases are generated in two steps: a) we generate full truth tables from CEMs; b) we search the truth tables to find a set of tests (entries in the table) that satisfy the required constraints (depending on the coverage criteria selected). In order to achieve the first step, we encode Boolean expressions as input of a SAT Solver (we use Sat4J [14]) and we produce all possible solutions for the expressions. The combination of all solutions forms the truth table. Truth tables are stored in memory using a Bitset representation for two reasons: first, it is compact, requiring less space in memory (important for large truth table of complex expressions); second, it allows the use of bitwise operations that speed up the search for a specific pattern.

#	X1	X2	X3	Z	Test pair		
					X1	X2	X3
1	0	0	0	0			
2	0	0	1	0			
3	0	1	0	0			
4	0	1	1	0	**8**		
5	1	0	0	0			
6	1	0	1	0		**8**	
7	1	1	0	0			**8**
8	1	1	1	1	**4**	**6**	**7**

Fig. 3: MC/DC test pairs for *Z = X1 and X2 and X3*. The condition X1 needs the two tests 4 and 8; X2 needs the tests 6 and 8; X3 needs the tests 7 and 8. The test set is {4,6,7,8}.

For the second step, we use a search method inspired by the filtering mechanism used in CAN message broadcasting [7]. The method requires configuring two objects ($filter$ and $mask$), each of which is a Bitset with the same size as the Bitsets generated for the truth table. The mask is used to determine which conditions are of interest and the filter is used to define what outcomes we want to vary. More precisely, we search for $source$ entries in the truth table such that

$$source \textbf{ bitwise-AND } mask = filter$$

For example, consider Figure 3 with the truth table of *Z = X1 and X2 and X3*. To find a pair of tests for condition X1 that satisfy the requirements of MC/DC, we use the mask 1110 and the filters 0110 and 1110. Intuitively, the mask denotes that the first 3 columns (X1, X2, and X3) are of interest; the two filters differ in the first component to satisfy **(R1)**.

D. Requirements and test cases tractability

REQIF (Requirements Interchange Format) [23] is a standard file format used to exchange requirements between software tools from different vendors. REQCAP has been implemented to enable users to import requirements directly from REQIF files provided by the client using a requirement management tool, such as IBM Rational DOORS [1]. Once imported, the user can assign one requirement (or a group of them) to the model elements (parent or child nodes) of DCs. This enables us to demonstrate the traceability between the original requirements and the generated test cases.

IV. EVALUATION

We used REQCAP to model part of an Oily Water Sewer (OWS) System, a level control system that maintains the level of a waste liquid in a sump tank using three discharging pumps. This is a real-world system that was developed and deployed by Applied Integration UK before REQCAP was created. In this section we discuss some benefits that REQCAP offers, when compared with the approach that was originally used.

The system has been implemented using a programmable logic controller (PLC) supporting 16 analogue inputs and outputs (AI and AO), and 16 digital inputs and outputs (DI and DO). The pumps can be controlled in three different modes of operation: automatic, remote manual, and local manual. In the *automatic* operation mode, the PLC monitors the level of

the oily water and controls the starting/stopping and the speed of the pumps to maintain the level setpoint. In the *remote manual* mode, the operator manually starts/stops the pumps and sets the speed from a distributed control system (DCS) console. Finally, in the *local manual* mode, the pumps can be started/stopped from a local pushbutton station at a fixed speed. Figure 4 depicts an overview of the OWS system.

In this paper we only discuss the modeling of the automatic mode.

A. Modeling the automatic mode

The requirements related to the automatic mode of operation are:

- When the level is below the "Pump Stop Level" all pumps shall be stopped.
- When the "Pump Start Level" is reached, the duty pump is started at its minimum speed. The PLC adjusts the speed to maintain the "Level Control Setpoint".
- If the pump reaches its "High Speed Start", the next duty pump is started and the PLC output is adjusted to split the load between the two pumps.
- If the speed falls to the "High Speed Stop" setting, the second pump is stopped.
- When the "Pump Stop Level" is reached, all pumps are stopped.
- The PLC will initiate high and low level alarm beacons local to the sump. This alarm will auto reset.

Following the approach described in the previous section, we started by developing the procedural view of the system using dependency charts (DCs). All low-level (green) nodes were then specified using cause-effect matrices (CEMs). We now show some extracts from our model, taken directly from REQCAP. For simplicity, we show a version where only two pumps are available.

a) Hierarchical dependency charts (DCs): Figure 5 shows four screenshots of REQCAP, each showing different abstract levels from the same model. In Figure 5a, the main window contains the root DC, which consists of the default Start and End nodes (representing the start and end of the automatic mode), and of a single high-level (blue) node representing the functional unit associated with the automatic mode

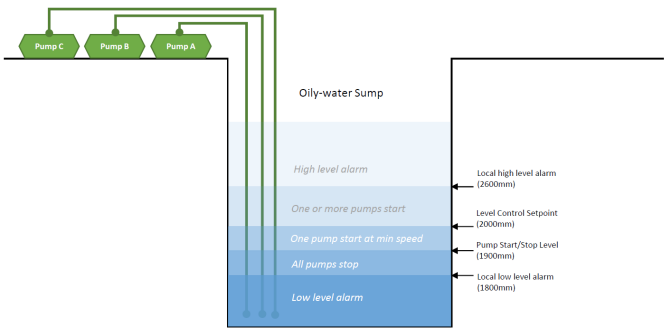


Fig. 4: Oily water sewer system overview

(*Auto Level Ctrl*). The left pane shows the *Model Explorer* with an expanded tree view of the model: we can see that the node *Auto Level Ctrl* is further decomposed into four high-level (blue nodes). These four nodes, labelled *High Level Ctrl*, *PID Speed Ctrl*, *Low Level Ctrl*, and *Alarm-Trip Ctrl* are shown in Figure 5b. These are also further refined into more atomic units: for example, the node *High Level Ctrl* is refined into the DC shown in Figure 5c and the node *Low Level Ctrl* is refined into the DC shown in Figure 5d.

b) Cause-effect matrices (CEMs): The DCs shown in Figures 5c and 5d contain only low-level (green) nodes that can be formally defined using CEMs. For example, we can define the node *Stop A and B* shown in Figure 5d with the cause-effect matrix shown in Figure 6. The matrix captures the requirement "*When the level is below the 'Pump Stop Level' all pumps shall be stopped.*", where the stop level is defined as *Level_7*.

B. SFC Generation

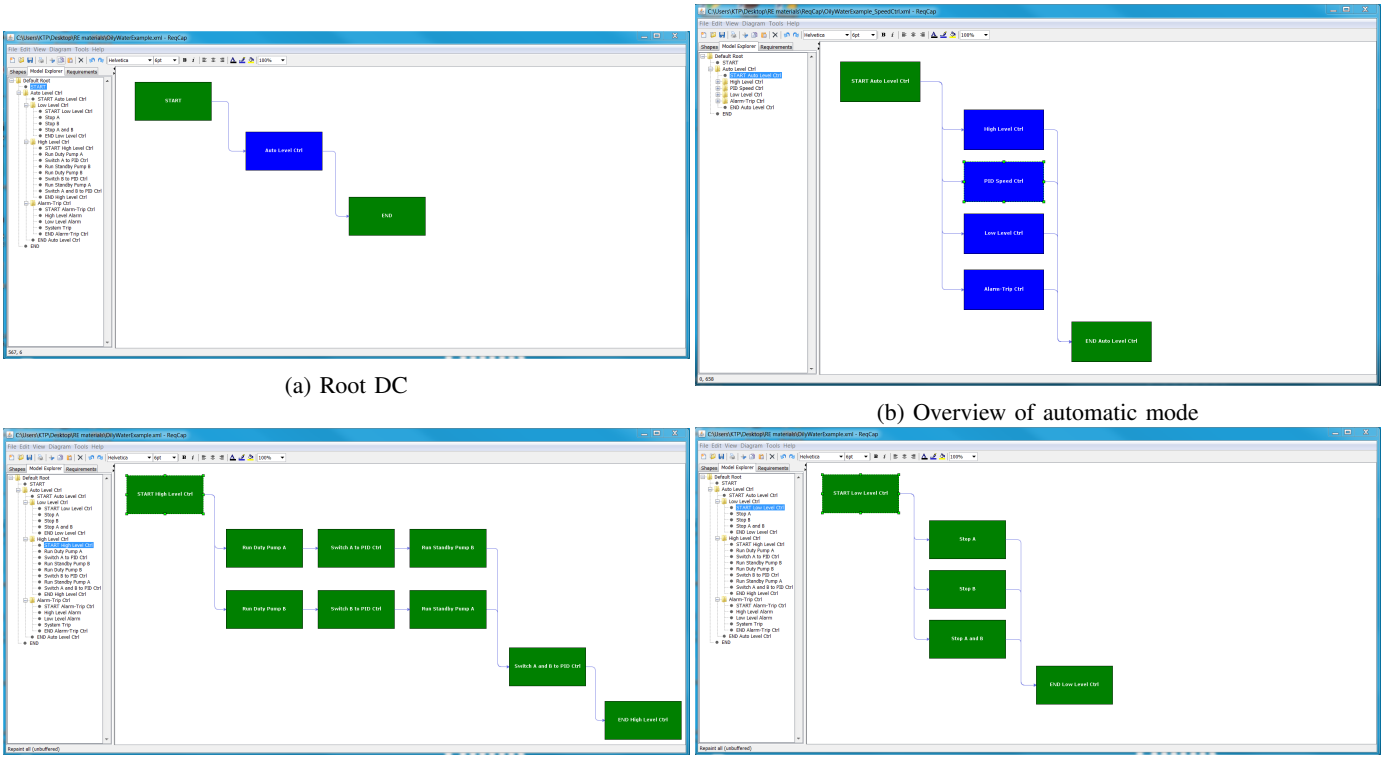
SFC models can be generated automatically at any stage of the system modeling. These can help control engineers and clients in visualizing and reviewing the system, since it is a well-established and popular domain-specific language. As an example, we show in Figure 1 the SFC model that REQCAP generated for the high-level control.

C. Test Case Generation

Test cases can be generated for the system (or part of it) from the cause effect matrices describing child (green) nodes. Users can adjust the number of the generated test cases according to the four standard testing coverage criteria listed in the previous section.

Figure 7 depicts the interface provided to select the testing coverage criteria. Note that users can select different coverage criteria for different transitions. The DC test coverage criteria generates the minimum number of tests, whereas MCC generates the maximum number of tests. MC/DC is chosen by default.

Table I shows some tests generated using REQCAP. The tool generated most of the tests listed in the original test document. However, using a first version of our model, tests 6, 7, 10, and 11 were not generated. We analyzed this issue and we found that these tests were missing because the part of the system describing the speed control behavior was not modeled completely. After completing the model, all tests were generated (see last column of the table). Additionally, the tool generated more tests than previously available in the original test document. This includes test cases that validate the system behavior when the conditions are not fully met or are invalid (rows 14 and 15 of Table I show two examples, but there are many more). This type of testing is called negative path testing. We have received very positive feedback from the engineers in the focus group on the tool capability of generating test cases covering both positive and negative paths, because it increases their confidence in the reliability of the systems they develop (note that, in practice, it is easier to miss negative path tests).



(a) Root DC

(b) Overview of automatic mode

(c) High-level control

(d) Low-level control

Fig. 5: Screenshots of REQCAP showing high-level views of the automatic mode model

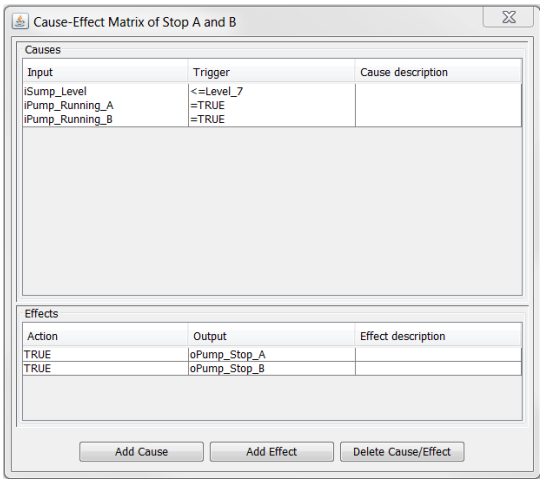


Fig. 6: Cause-effect matrix associated with the node *Stop A and B* in the low-level control (see Figure 5d)

D. Discussion

REQCAP puts together a set of features that support engineers (and clients) in visualizing, reviewing, and reusing requirements. First, the model-driven approach supported by REQCAP pushes the user to establish a good understanding of the system's behavior before implementation. Second, the DCs and all the artefacts generated by REQCAP can be shown to clients to clarify requirements and remove ambiguities. Feedback is immediate: if during a meeting with a client, a change to the model is proposed, the DCs, SFCs, and test cases can be immediately generated to provide different views on the proposed changes. Third, the capacity to generate test cases according to different coverage criteria introduces a pragmatic approach to increase reliability. This feature is particularly interesting in the context of safety-critical systems, where often certain code coverage criteria have to be followed (e.g. ISO 61508 may require MC/DC code coverage, depending on the SIL level that needs to be achieved). The negative path testing illustrated in the case study above is another benefit of using REQCAP. Finally, REQCAP opens the possibility to build libraries of components (i.e. a repository of designs) that can be reused.

In summary, we believe that the results achieved so far demonstrate that REQCAP can be used to effectively capture requirements for realistic industrial control systems and can generate valid and useful test cases.

V. Related Work

Several tools have been proposed for the enrichment of the modeling process of industrial control systems. Here, we focus our attention on tools that support graphical models.

A work closely related to the one presented here is [17], which introduces a systematic procedure for the design of logic controllers. The procedure uses intermediate formats to transform informal requirements into an SFC. Informal requirements are initially formalized using DCs (similar to

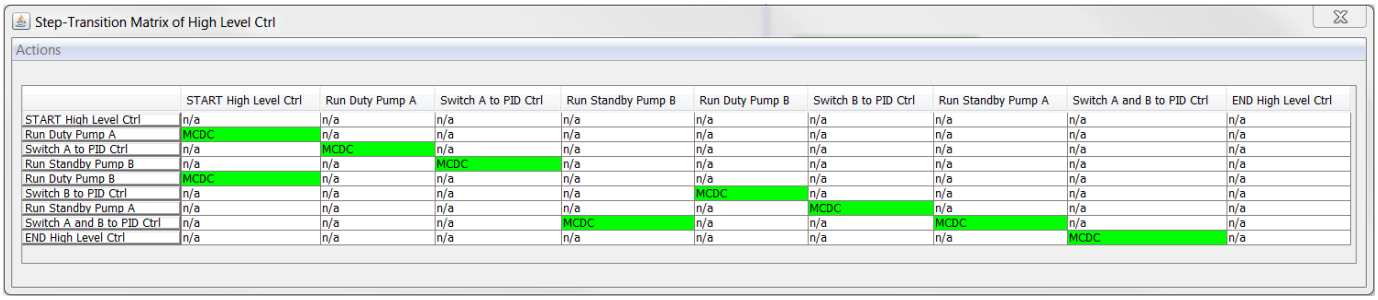


Fig. 7: Interface to select the testing coverage criteria. Users can select different coverage criteria for different transitions (default is MC/DC).

ID	Test Description	Original Test	Generated Test (#1)	Generated Test (#2)
1	Pump A (duty) starts correctly when the level > Level 8	yes	yes	yes
2	Pump A (duty) stops correctly when the level < Level 7	yes	yes	yes
3	Pump B becomes Duty and starts correctly when the level > Level 8	yes	yes	yes
4	Pump B (duty) stops correctly when the level < Level 7	yes	yes	yes
5	Pump A now becomes duty and starts correctly when the level > Level 8	yes	yes	yes
6	**PID speed control of pump A is enabled when the level has continued to rise between Level 7 and Level 8**	yes	**no**	yes
7	**PID speed control of pump A will ramp speed down to minimum when the level is decreased between Level 7 and Level 8**	yes	**no**	yes
8	Pump A (duty) stops correctly when the level < Level 7 whilst in PID mode	yes	yes	yes
9	Pump B becomes duty and starts correctly when the level > Level 8	yes	yes	yes
10	**PID speed control of pump B is enabled when the level has continued to rise between Level 7 and Level 8**	yes	**no**	yes
11	**PID speed control of pump B will ramp speed down to minimum when the level is decreased between Level 7 and Level 8**	yes	**no**	yes
12	High level alarm is activated and PID speed output continues to rise when the level is increased to Level 9 > Level 8	yes	yes	yes
13	High level alarm is deactivated and pump A and B stop when the Level is decreased to Level 7	yes	yes	yes
14	If both pumps are stopped and the level < Level 7, keep pump A stopped	**no**	yes	yes
15	If both pumps are stopped and the level < Level 7, keep pump B stopped	**no**	yes	yes

TABLE I: Tests generated by REQCAP. The first two columns identify the test by ID and description. The third column states whether the test was present in the original test document. The fourth and fifth columns state whether the test was generated by REQCAP from the incomplete and complete model, respectively.

REQCAP) and function tables (FTs), which are iteratively refined and subsequently translated into a function plan (FUP) — representing the structure of the SFC, but without action blocks and transition conditions — and a step-transition chain (STC) — which consists of the transitions, steps, and action blocks. The final step is to combine the FUP and STC to generate the logic controller as an SFC. The SFC is created for the purpose of formal verification via model checking. Our work, on the other hand, is focused on generating high-level SFCs suitable for reviewing the system and for collecting feedback from clients. Moreover, test cases satisfying a number of standard test coverage criteria are also generated.

In [8] the informal requirements are also formalized in an iterative refinement process but now the resulting DC/FT data structure is stored in a generic XML data format from which DC/FT and SFC can be generated without translation. This ensures consistency between the DC/FT and the SFC and preserves readability and maintainability.

In [11] and [12] a model-based approach to PLC software development is introduced. Commonly used methods view control systems as regards to the physical building blocks (the equipment) but, similarly to our work, the authors propose a procedural view. They introduce a domain-specific modelling language called ProcGraph [9] that is specialized for the domain of procedural process control software. ProcGraph has been supported with the creation of a tool, Gecko, which facilitates the creation of graphical models, automatic IEC 61131-3 code generation, and automatic generation of documentation skeletons. ProcGraph has also been used to support model-driven development of industrial process control software [18].

In [6], a model-based test suite generation method is presented. The method converts FBD programs into timed automata and uses model checking to generate tests. The most important difference between [6] and our work is that REQCAP was developed to assist in the process of capturing requirements highlighting the procedural nature of control systems. From the captured requirements, it allows the generation of tests, but also the generation of SFCs. It would not be difficult to extend REQCAP to also generate FBD programs that could then serve as input to the work reported in [6]. This difference also applies to the work described in [24], where the authors present an algorithm that constructs test sequences from Mealy machines.

VI. Conclusion

We present ReqCap, a model-based requirement modeling tool that implements a new method that can be used to effectively capture requirements of PLC-based systems. It supports automated generation of test cases according to different coverage criteria, it can import requirements directly from ReqIF files, and it is capable of automatically generating Sequential Function Charts (SFCs). We use a real-world case study to show how ReqCap can be used to model realistic system requirements. We discuss how the automated generation of SFCs and test cases can support engineers (and clients) in visualizing and reviewing requirements. All the tests listed in the original test document of the case study are generated automatically by ReqCap. Moreover, ReqCap generates *negative path* tests that are not in the original test document. This demonstrates that ReqCap can be used to effectively capture requirements and generate valid, useful, and thorough test cases.

This work was developed in an industrial context and captures best practice gathered by years of experience delivering real PLC-based projects. We believe that ReqCap addresses the three challenges outlined in Section II: it can be used to standardize and unify modeling and development methods, it introduces automation by generating SFCs and good-quality test cases, thus reducing costs and increasing consistency across projects. In fact, ReqCap is now being used on a safety-critical live project that needs to comply with the functional safety standard IEC 61508. The MC/DC test coverage will be a requirement, so ReqCap's support is envisaged to be of great help.

We are currently in the final implementation stage of a test automation tool that can import the generated test cases from ReqCap and communicate with different types of PLCs to run the tests regardless of the PLC manufacturer (e.g., Siemens, Rockwell, or ABB). This will allow us to run an exhaustive-type of testing (when using MCC coverage, for example) on the target machine (i.e., the PLC unit under test). This approach will deliver the same level of confidence of fully formal approaches such as model-checking methods, but with the advantage that the tests are run on a real machine rather than executed symbolically using abstract models. In the future, we plan to use existing formal models (e.g. [2], [3]) to formally verify certain aspects of ReqCap, such as the algorithm that generates the SFCs and the test generation algorithms. In terms of user interaction, we plan to investigate the suitability of pen-based devices to facilitate the creation of hierarchical dependency charts (by reusing material developed in [20]).

References

[1] IBM Rational DOORS. http://www-03.ibm.com/software/products/en/ratidoor. Accessed: 2017-02-17.

[2] Jan Olaf Blech and Sidi Ould Biha. Verification of PLC properties based on formal semantics in Coq. In *International Conference on Software Engineering and Formal Methods*, pages 58–73. Springer, 2011.

[3] Jan Olaf Blech and Sidi Ould Biha. On formal reasoning on the semantics of PLC using Coq. *arXiv preprint arXiv:1301.3047*, 2013.

[4] SW Cheon, JS Lee, KC Kwon, DH Kim, and H Kim. The software verification and validation process for a PLC-based engineered safety features-component control system in nuclear power plants. In *Industrial Electronics Society (IECON)*, volume 1, pages 827–831. IEEE, 2004.

[5] International Electrotechnical Commission. International standard CEI IEC 1131-3, Programmable Controllers—Part 3: Programming languages. 1993.

[6] Eduard Paul Enoiu, Daniel Sundmark, and Paul Pettersson. Model-based test suite generation for function block diagrams using the uppaal model checker. In *Proceedings of the Software Testing, Verification and Validation Workshops*, pages 158–167. IEEE, 2013.

[7] Mohammad Farsi, Karl Ratcliff, and Manuel Barbosa. An overview of controller area network. *Computing & Control Engineering Journal*, 10(3):113–120, 1999.

[8] Stephan Fischer, Martin Hfner, Christian Sonntag, and Sebastian Engell. Systematic generation of logic controllers in a model-based multi-formalism design environment. {*IFAC*} *Proceedings Volumes*, 44(1):12490 – 12495, 2011. 18th {IFAC} World Congress.

[9] Giovanni Godena. Procgraph: a procedure-oriented graphical notation for process-control software specification. *Control Engineering Practice*, 12(1):99–111, 2004.

[10] Bogdan Humoreanu and Ioan Nascu. Wastewater treatment plant scada application. In *Automation Quality and Testing Robotics (AQTR)*, pages 575–580. IEEE, 2012.

[11] Gregor Kandare, Giovanni Godena, and Stanko Strmčnik. A new approach to PLC software design. *ISA transactions*, 42(2):279–288, 2003.

[12] Gregor Kandare, Stanislav Strmčnik, and Giovanni Godena. Domain specific model-based development of software for programmable logic controllers. *Computers in Industry*, 61(5):419–431, 2010.

[13] Hayhurst Kelly J., Veerhusen Dan S., Chilenski John J., and Rierson Leanna K. A practical tutorial on modified condition/decision coverage. Technical report, 2001.

[14] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[15] Jong-Hoon Lee and Junbeom Yoo. Nude: Development environment for safety-critical software of nuclear power plant. In *Transactions of the Korean Nuclear Society Spring Meeting*, volume 2012, pages 1154–1155, 2012.

[16] Robert W Lewis. *Programming industrial control systems using IEC 1131-3*. Number 50. IET, 1998.

[17] Sven Lohmann and Sebastian Engell. Systematic logic controller design as sequential function chart starting from informal specifications. *Chinese Journal of Chemical Engineering*, 16(1):43–47, 2008.

[18] Tomaž Lukman, Giovanni Godena, Jeff Gray, and Stanko Strmčnik. Model-driven engineering of industrial process control applications. In *Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2010.

[19] R Mandal, A Kumar, TMG Kingson, RK Pd Verma A Kumar, S Dutta, SK Chaulya, and GM Prasad. Application of programmable logic controller for gases monitoring in underground coal mines. *IRACSTEngineering Science and Technology: An International J.(ESTIJ)*, 3(3):516–522, 2013.

[20] Alexandra Mendes, Roland Backhouse, and Joao F Ferreira. Structure editing of handwritten mathematics: Improving the computer support for the calculational method. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*. ACM, 2014.

[21] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[22] Khenaidoo Nursimulu and Robert L. Probert. Cause-effect graphing analysis and validation of requirements. In *Proceedings of the 1995 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '95, pages 46–. IBM Press, 1995.

[23] OMG. Requirements Interchange Format (ReqIF). Technical report, July 2016.

[24] Julien Provost, Jean-Marc Roussel, and Jean-Marc Faure. Generation of single input change test sequences for conformance test of programmable logic controllers. *IEEE Transactions on Industrial Informatics*, 10(3):1696–1704, 2014.

[25] Detlef Streitferdt, Georg Wendt, Philipp Nenninger, Alexander Nyßen, and Horst Lichter. Model driven development challenges in the automation domain. In *Computer Software and Applications (COMPSAC)*, pages 1372–1375. IEEE, 2008.