Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Model-based testing of PLC software: test of plants' reliability by using fault injection on component level

Susanne Rösch * **Dmitry Tikhonov** * **Daniel Schütz** *
Birgit Vogel-Heuser *

* *Technische Universität München, Institute of Automation and Information Systems, 85748 Garching bei München, Germany (e-mail: {roesch; tikhonov; schuetz; vogel-heuser}@ ais.mw.tum.de).*

Abstract: In this paper, the current situation of how PLC software is tested in industry is analyzed and the challenges on new testing approaches are identified using real industry code and a survey conducted within industry. The different possible and most relevant faults that may occur and must be dealt with are identified and requirements for testing approaches concerning component failures are derived. Further on, an approach to generate tests for error handling routines, which test the reliability of plants by injecting the corresponding faults is presented. The test cases are generated from timing sequence diagrams in combination with failure mode and effects analysis. In order to inject the faults at relevant points during the execution of the control software, IEC 61131-3 code is analyzed for the derivation of the test cases.

Keywords: plant automation, test generation, model-based development, programmable logic controllers, reliability, software engineering, manufacturing systems, IEC 61131-3

1. INTRODUCTION

Today's Programmable logic controller (PLC) application software development in plant engineering poses increasing challenges as more and more functionality of plants is realized through software (Reimann (2013)). At the same time, high requirements on robustness and reliability of these plants must be met and verified. A plant's robustness and reliability are especially a result from its correct reaction to unexpected events. This means that a plant reacts to the unexpected with a suitable strategy, which must be implemented in error handling routines. These routines must be implemented correctly and at the correct places in the PLC software. In order to ensure the correct functionality of these software error handling routines, special strategies to test them must be developed, as they are not automatically executed during the normal execution of the software, when no faults occur. In order to test these reactions of plants to the unexpected, i.e. faults occurring within the system or its environment, a testing approach using fault injection (FI), and hence, testing the error handling routines is presented in this paper. For the FI process, common faults that may occur within a plant are feigned while its reaction is monitored to verify the correct execution of the designated error handling routine within the application. To prove the reliability of safety critical plants, conformance tests, which test the conformance of systems' to their specification, i.e. if they are correctly executed on the target platform (Provost et al. (2011)), are necessary. In this paper the subject of conformance testing is not focused but rather only the error handling routines implemented within the application controlling the plant are focused. The paper is structured as follows: In the next section, requirements on a testing approach in the domain of plant automation are analyzed and the most relevant faults that must be dealt with are identified based on a survey and interviews conducted within industry and an analysis of industry PLC code. Subsequently, related work is presented. The concept to generate test cases that use FIs to test the error handling routines within the software of the plant as well as the results from a first prototypical implementation are illustrated using a small laboratory plant in section 4 and 5. The paper is concluded by a summary and an outlook on future work.

2. REQUIREMENTS ANALYSIS BASED ON SURVEY AND INTERVIEWS IN INDUSTRY

In order to analyze and evaluate the requirements on a test case generation approach three actions were taken. Firstly, a survey was conducted with 8 software engineering experts from 7 different companies in the field of plant automation. Secondly, following the survey, an expert discussion and interviews with the participants of the survey were lead. Finally, real industry code was analyzed to refine the identified requirements. The survey targeted, on the one hand, current industry practice of testing plants to derive which requirements must be met to integrate a new approach smoothly into current development processes. On the other hand, the relevance of different aspects (0%: low relevance, 100%: very high relevance) such as

⋆ The IGF-project 16906 N of the research association electrical engineering of the ZVEI e.V. was sponsored via the AiF as part of the program to support cooperative industrial research (IGF), with funds from the Federal Ministry of Economics and Technology (BMWi) following an Order by the German Federal Parliament.

the occurrence of different types of faults was asked. The four main requirements (*R1- R4*) that were identified are explained in the following.

In current practice, not much time is explicitly reserved for testing (25%-30% of the time in the software development process) and simulation models cannot be used most of the time. It was rated that a maximum of 20% of the development time is acceptable for implementing such a model. This is not realistic in plant automation as plants are often customized. Suitable models for these individual plants are complex to realize and the customers do not pay for the development of such a model. In conclusion *it must be possible to test against the real plant as well as different prototypes or partial solutions (relevance 91%) which are available during the different development stages as well as against simulation (R1).*

Several questions of the survey targeted at identifying suitable specifications that are commonly available before software implementation and therefore for test case generation in the development process. The results showed a small variety of specifications such as requirements tables, timing diagrams (TDs), high-level state charts, textual pseudo-code and failure mode and effects analysis (FMEA). Several of these specifications are applied before the implementation of the software since each specification is suited best for a specific engineering domain and different aspects of a plant. TDs and state charts are used to specify the expected behavior during normal (mainly automatic mode) operation. In the expert discussion it was rated that TDs are used most of the time to model complex behavior of the plant. Different forms of TDs do currently exist. In order to make test case generation possible, the requirements on the TD specification were refined based on the analysis of PLC code from four different of the above mentioned companies.

The TD used to specify behavior for a test generation approach in plant automation needs to include elements for depicting time constraints, value lifelines which correspond and represent the states of the components and messages between the different components. Value lifelines are especially needed as they can represent the I/Os which are linked to the software and represent the current state of a plant from the software point of view. Another important aspect is that continuous value lifelines are needed, because many values within the software, e.g. the pressure in a cylinder, follow continuous courses.

In order to allow automatic generation, the diagram needs to be formalized. The different operating modes, e.g. automatic, manual, step chain mode, etc. in plant automation are another aspect that must be dealt with when testing as some alarms and interlockings are necessary and specified in one mode while they are not allowed in another. In conclusion the test case generation shall be done by *extraction of information from TDs, which are formalized and include value lifelines which represent the state of the components and their interactions (R2a).*

The analysis of control code and subsequent interviews with the software developers on how error handling routines are implemented in current practice revealed that each error, fault or failure must be acknowledged or recorded by a specified message. The expected behavior, e.g. going into a safe state when faults occur, is not specified for each single fault, but it is determined by the fault class predefined by the developers. One fault class, e.g., is that the fault is only reported by a warning, other faults belonging to a different fault class must be handled by shutting the plant down immediately.

As there is little time and many tests during commissioning cannot be conducted without surveillance of the plant, the scalability of future testing approaches is a crucial success factor. As the object-oriented paradigm which has been introduced with the 3rd edition of the IEC 61131-3 standard might improve the testing effort and reduce the number of test cases that have to be conducted is not yet fully introduced in industry, other means of choosing the test cases are proposed in this paper. As a resulting requirement it is postulated that *fault classes and the means to prioritize test cases are to be included in the specification (R2b).*

To determine which faults are common and therefore need to be focused on and tested in plant automation, the participants were asked to name typical and critical faults in the survey. The faults named were: component failures (relevance 60%), process faults (56%), mechanical faults (54%), software faults (46%) and faulty operator behavior (44%). Some of the faults were further refined in the following discussions, e.g. process faults include collision, wrong timing, etc.; mechanical faults include abrasion, wearout, etc.; software faults include division by 0, memory access, etc. As each of these faults has different causes, each of them must be analyzed individually and dealt with accordingly. In this paper we focus on component failures, which are tested by injecting faults which represent them. Therefore *FI on component level must be explicitly incorporated in the approach (R3).*

Another aspect that was discussed was the visualization of test cases. A *comprehensible visualization and the option to modify generated test cases (R4)* was emphasized by the industry experts in order to support test engineers in understanding the generated test cases and to enable maintainability and proper documentation.

3. RELATED WORK

In this section, firstly related work is discussed regarding the specification of TDs which are to be used according to *R2a* as the basis for test case generation. Following this, other approaches for systematic test generation in plant automation and FI are discussed.

3.1 Formalization of TDs

Several works such as Vyatkin and Bouzon (2008) and Katzke and Vogel-Heuser (2007) have formalized TDs using some form of timed automata. TDs are formalized using net condition/event systems (NCES) by Vyatkin and Bouzon (2008). By implementing such a transformation, a formal verification of the specified behavior is done and it can be checked if the specified behavior of the TDs matches one or more given paths within function blocks. However, in the TD specification used, the focus lies on the verification of condition operators specified between the state changes of the lifelines. Interactions between the

lifelines are not modeled (*R2a*). Formal verification is also focused on by Preusse and Hanisch (2008). Computation Tree Logic formulas are automatically derived from the formally specified symbolic TD. These are verified using model-checking techniques against NCES models as well. The symbolic TD represents sequences of states where simultaneous transitions can be explicitly expressed. Real-time conditions have been excluded in this work. In the formal specification of real-time symbolic TDs, this element has been added by Feyerabend and Josko (1997), in order to express those constraints formally. Interactions and continuous value changes can not be modeled in the symbolic TD specifications (*R2a*).

On the other hand there is the Unified Modeling Language for process automation (UML-PA) (Katzke and Vogel-Heuser (2007)), which adapts and defines the TD and sequence diagram (SD) into the timing sequence diagram (TSD) especially for the domain of plant automation. Using the UML-PA TSD, life lines, state life lines, which represent the state of an object, and messages as specified in the UML SD between the different lifelines can be modeled. By using the port element even interfaces may be depicted. In the UML-PA, using state lifelines, only discrete time intervals are considered, and the I/O image cannot be represented as it is given in by the process. The TSD is formalized based on a representation of state charts with an extension by Timers. The states refer to the state of a lifeline in the TSD, events define operations and refer to the messages in the TSD, sets of transitions correspond to a flank (i.e. change of value on a lifeline), and the initial state and the final state correspond to the initial values in the TSD (Katzke and Vogel-Heuser (2007)). As the UML-PA TSD fulfills many but not all of the requirements concerning *R2a*, an adaption of this diagram is used in the approach presented in this paper.

3.2 Systematic test case generation

In the fields of model based testing and systematic generation of test cases from specifications a lot of research has been conducted in recent years. Krause et al. (2008) automatically generate test cases from formalized UML state charts based one a transformation to Extended Safe Place/ Transition Nets. In doing this they can automatically generate test cases reaching full path coverage as well as apply formal verification strategies from petri net theory. A visual representation of the test cases is done using the Testing and Test Control Notation (TTCN-3) (Baker et al. (2004)). UML diagrams are used in the work of Kumar et al. (2011) and Hametner et al. (2010) as well. In Kumar et al. (2011) test cases are derived from UML state charts and represented in the TTCN-3 notation. The set of test cases is reduced by an all transition coverage. The approach could be proven to work for simple communication systems, but an extension for testing PLCs is another goal of the research group. Hametner et al. (2010) identify the UML models which are best suited to extract test cases. One diagram which is identified as suitable to directly derive test cases, is the TD of the UML. A first example shows how to derive test cases from state charts and SDs. A formalization of UML diagrams has also been conducted by Basile et al. (2009), which may be used for test case generation or verification. Hussain and Frey (2006) propose the derivation of test cases from UML diagrams as well. The generation of test cases is done through path analysis of behavioral diagrams especially for applications of IEC 61499.

All of the approaches described in this section do neither take FI, which needs to be explicitly regarded to execute the tests on the real plant (*R1*, *R3*), into consideration. Nor were the requirements for these approaches taken from plant automation industry examples.

3.3 FI in reactive systems

FI approaches can be divided into hardware-implemented FI, where faults are for example injected by forcing pins, software-implemented FI, where faults of the system are emulated by the software or FI in models (Svenningsson et al. (2010)). While FI is rarely done in the field of plant automation it is already established in the automotive domain. Svenningsson et al. (2010) and Schlingloff and Vulinovic (2005) present approaches, where the behavior of the system in reaction to various different kinds of faults can be tested using Simulink models. This approach is also suitable for emulating component failures but it is focused on the simulation environment and not developed in order to meet PLC platform requirements (*R1*). In a previous work of this research group a model-implemented FI approach for plant automation has been implemented using a UML state chart simulation environment (Kormann and Vogel-Heuser (2011)). The FI depends on reaching a predefined state of the simulation, and thus a software-implemented FI is not possible (*R1*).

4. CONCEPT FOR TESTING ERROR ROUTINES

In this section firstly the application example that shall illustrate the concept in the following is introduced. Further on, an overview on the concept is given and afterwards the test case generation process is explained in detail with special emphasis on what information is extracted from which source.

4.1 Application Example

The concept is explained and evaluated using a laboratory plant which sorts and stamps work pieces (WPs) (Legat et al. (2013)). The plant is composed of several modules. The stamping module is modeled in detail to illustrate the approach in this paper (Fig. 2, right). The module has a slider which accepts WPs and a stamp cylinder which stamps the WPs at a defined pressure. The stamp cylinder includes three sensors: two end position sensors *StampUp* and *StampLowered*, and a pressure sensor. The slider includes three sensors as well: one to detect WPs (sensor *Filled*) and two end position sensors of the cylinder. The end position sensor *PosStamp* indicates, that the WPs is at the position of the stamp and thus can be stamped, *PosCrane* indicates whether the slider is in the position to receive or give away WPs to the next interacting module. An overview of the implementation of the module is shown in Fig. 2 on the bottom left in the IEC 61131-3 function block diagram (FBD) language.

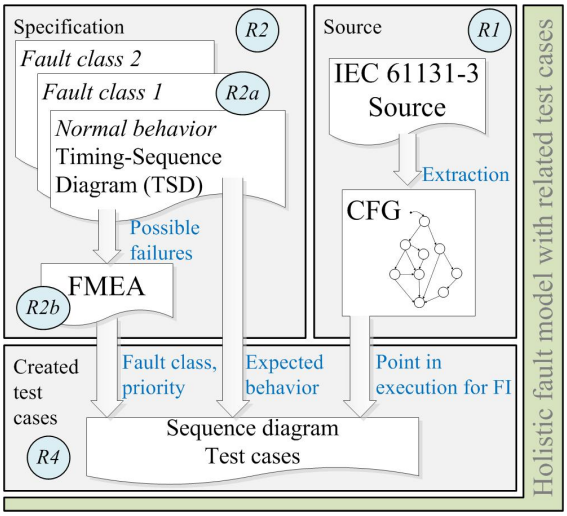

Fig. 1. Overall concept for test case generation

4.2 Concept overview

The overall concept for generating test cases to test error handling routines is shown in Fig. 1 with the requirements defined in section 2 connected to them. The expected behavior during normal operation of the system under test (SUT) and the behavior in reaction to an occurrence of faults from each fault class is specified using adapted TSDs. Possible failures are derived by assuming that each deviation from the normal behavior is a fault and needs to be handled by an error handling routine. This information, i.e. deviation of the normal behavior, is transferred into an adapted FMEA. Then the fault class is specified, which is added as an additional column in the FMEA. It determines which behavior that was specified in the fault class TSDs is expected in the error handling routine that is to be tested.

The only information then missing is which execution path corresponds to the behavior defined in the TSD. The Precondition defined in the TSD is extracted and added as the initial condition to the test case. By extracting and analyzing a control flow graph (CFG) of the PLC program the path which corresponds to the Precondition and the Postcondition can be identified. When the Precondition is fulfilled during the execution of the program, the test case for the defined component failure is started. By modeling the normal behavior and the behavior in reaction to an occurrence of faults from different fault classes in combination with the information from the FMEA and the generated test cases, a holistic model of possible faults and their related test cases is achieved. In the following the test generation process is explained in detail.

4.3 Extraction of information from TSDs

The modules of the laboratory plant were modeled according to an adapted UML-PA TSD notation as shown in Fig. 2 on the right. The model is mapped to the software structure of the PLC program as shown in Fig. 2. The module has two variables to communicate with other modules: *WorkpieceReady*, which signals the readiness of a WP to take it up by another module, and *StationEmpty*, which signals the readiness to receive a new WP. To represent the global variables which are mainly the sensor variables of the module and additionally the operating mode, lifelines are used. The lifelines depicted are connected to one or more sensors the stamping module owns through their value lifelines (e.g. Fig. 2, left, *Slider Position* is defined by two end position sensors: *PosStamp* and *PosCrane*). From the value lifeline in Fig. 2 it can be seen, that the value change is continuous, as the change does not happen discretely in the real process as well.

The messages between the different value lifelines signify operations with their corresponding actuator variables that initiate or must follow certain signals or values representing the state of the component. In this diagram as soon as a WP is delivered (Fig. 2, right, value lifeline *Filled*) it must be transported to the stamping position (*PosStamp*) within 0.5 seconds. Afterwards, the stamping process may take between 1 to 3 seconds. The TSD in Fig. 2 depicts the expected behavior of the system if no faults occur.

As can be seen in Fig. 2, different components are involved in the stamping process, and therefore have to be tested. The reactions, i.e. changes of value lifelines, to other changes on value lifelines are expected in a certain sequence and after a certain time within the sequence, which is expressed by the time constraints. Two of these reactions are highlighted by the lines Precondition and Postcondition in Fig. 2 on the right. The operation *Stamp(...)* is initiated as soon as the Precondition is met, which corresponds to the values representing the state that the module holds at this point in time. Then, firstly, the value change of *Stamp Position* and secondly, the value change of *Stamp Pressure* is expected. In this example they are marked by the line Postcondition in Fig. 2. For each of these value transitions one test is generated. The test checks what happens when the Postcondition is not met, which means that one of the expected value transitions does not happen in the specified time (see also extracted information from TSD in Fig. 3). In other words, the PLC will not be able to observe the expected value and needs to react accordingly with an error handling routine. In this case one test checks whether a missing value change after 0.5 seconds of *Stamp Position* is detected and an error handling routine executed, one test checks whether a missing value change of *Stamp Pressure* is detected. If a message (i.e. operation) with a sender and a related receiver are included in the time sequence, they are extracted as the initiator for the timing condition in the test case. This is the case in the depicted example, where *Stamp(...)* is the initiator for the time condition of 0.5 seconds. If no message is included, as for example the expected value change after 3 seconds of the stamping process, the timing condition is started as soon as the previous value change is detected. The operating mode is included as a lifeline in the TSD (see Fig. 2) and therefore is used in addition to define the Precondition (*R2a*). Theoretically a test could be generated for the other lifelines as well, i.e. a fault injection feigning a value change where no value change should occur. This is neglected right now and will be evaluated in future work.

4.4 Extraction of information from FMEA

As the expected reactions of a system and thus the Preconditions and the expected Postconditions for a test case

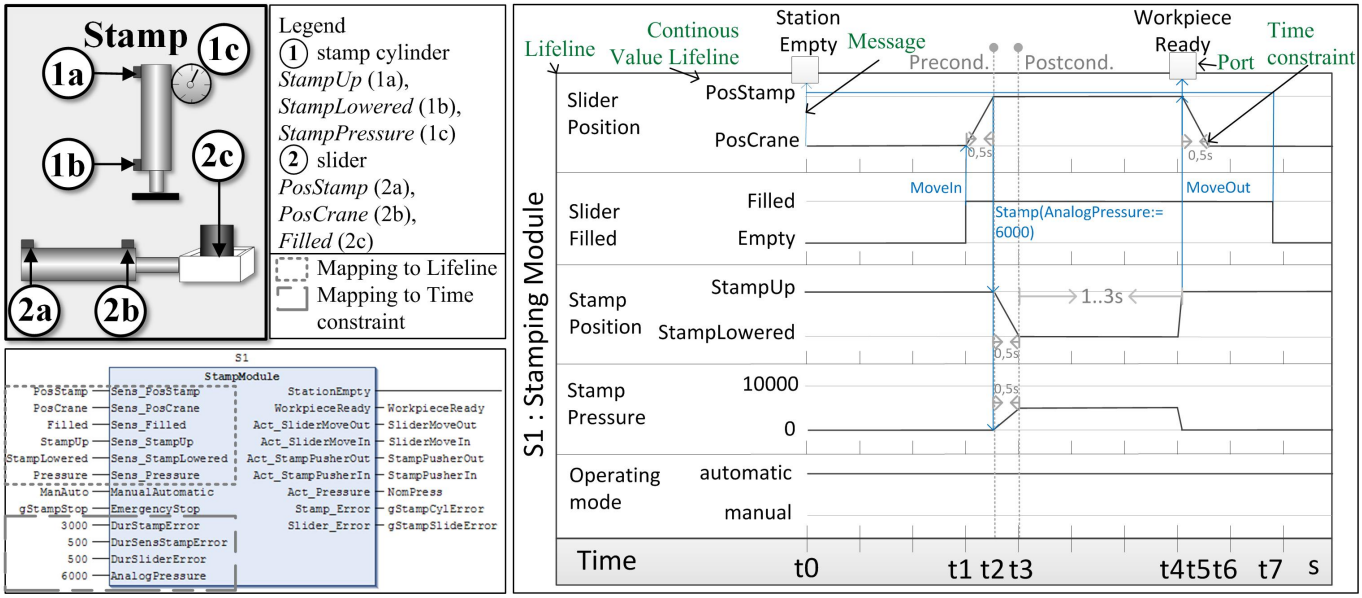

Fig. 2. Left: Sketch and software in FBD language of stamping module. Right: TSD of the stamping module

can be extracted from the TSD now, the only information missing is the priority of these test cases, the fault class to determine the expected behavior during the execution of the error handling routine and the alarm message to report the according fault. This is done using an adapted FMEA analysis to identify the most risky components (*R2b*). The information from the TSDs is extracted into the first column of the adapted FMEA analysis (Fig. 3) and is planned to be extracted automatically in the future. In the companies that were analyzed the FMEA is usually filled out by the electrical and mechanical engineers manually. This is done in the FMEA which is proposed here as well, because these engineers have the knowledge on the likeliness of the specified component failures but it is also used by the software engineers to test their software. In the FMEA, the column "S" signifies the severity of the failure, "O" stands for the occurrence of the failure and "D" rates how likely it is that the failure is detected. The calculated product of these three figures represents the risk priority number (RPN) of the failure and therefore corresponds to the priority of the test case. The other columns are explanatory and can be used as a means for documentation and discussion between the different domains later on. A reduced FMEA showing the most important aspects in Fig. 3, depicts the scenario of the operation *Stamp(AnalogPressure:=6000)*, and the following reaction of the stamp which fails to move out (value lifeline *StampLowered*) and therefore does not meet the expected behavior specified in Fig. 2. Another column is added to specify the fault class of the fault (alarm message in brackets) and therefore determines the expected reaction of the plant if this fault occurs. For the test case generation the RPN and the fault class are extracted in order to prioritize the test cases and assign the TSD where the reaction of the stamping module in reaction to the specified fault class is specified. Additionally, the alarm message which is expected for the according fault is extracted. Another possible source to extract this information are alarm lists, if they are assigned correctly to the corresponding faults.

System element: Stamping Module
Function: stamping

Information from TSD	Potential failure mode	S	Potential effect(s) of failure	Potential cause(s) of failure	O	Measures of detection	D	RPN	Fault class
NOT *StampLowered* after Precond,: max. 0.5 s Precond.: (Stamp(), PosStamp, Filled, StampUp, StampPressure == 0, automatic)	Cylinder cannot extract, Wrong stamping duration	5	WP is not stamped, WP is stamped too long, WP defective (gStampCyl Error)	Wrong position of WP, pneumatic mal-function, Sensor end position does not work	3	Check time condition	3	45	1 (Msg: gStamp Cyl Error)
Stamp Pressure **NOT** 6000 after Precond.: max. 0.5 s Precond.: (Stamp(), PosStamp, Filled, StampUp, StampPressure == 0, automatic)	Wrong stamping duration	5	WP is stamped too long, WP defective	Sensor end position or sensor pressure does not work, pneumatic malfunction	3	Check time condition, check air pressure	3	45	1 (Msg: gStamp Cyl Error)

Fig. 3. Excerpt of the FMEA of the stamping module

4.5 Extraction of information from IEC 61131-3 source

The ideal goal is that each feasible execution scenario of the control program, where the failure of the selected sensor may have a direct impact on the program behavior, is tested in order to achieve full test coverage. If the model of the TSD is complete, the TSD represents one of these paths. For the generation of the test cases a CFG of the source code of the stamp module is created extracting an abstract syntax tree beforehand as also implemented by Prähofer et al. (2012) and Biallas et al. (2012) for the purpose of static analysis. Using control and data flow analysis techniques the CFG path that corresponds to the test case is extracted from the model (in this example: *StampLowered* after *PosStamp*, see also Fig. 3). In the example two CFG paths were found that correspond to the TSD scenario. In order to test both scenarios an additional variable defining the respective control path is added to the Precondition of these to test cases. Another scenario of an initialization where the stamp is lowered, which was specified in another TSD and only varies in one value on the lifeline compared to the one in Fig. 2 corresponds to another path that needs to be tested for the failure of *StampLowered* and completes the test set for *StampLowered*.

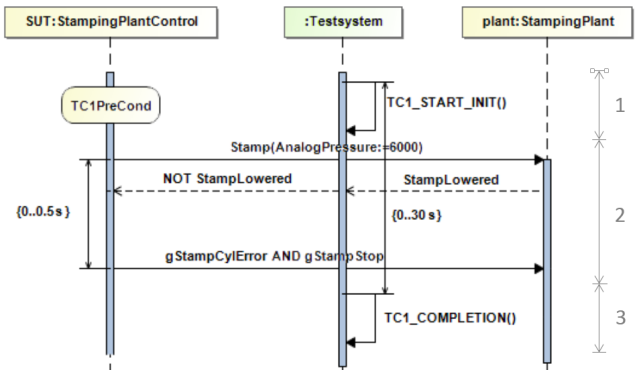

Fig. 4. Test set specification for FI for *StampLowered*

4.6 Generation and visualization of test cases

The test cases need to be visualized to support the test engineers in understanding and adapting them (*R4*). The SD of the UML is especially suited for this task because it shows the relevant test components and their respective black-box behavior, i.e. expected inputs and outputs as a message, and has also been established in the U2TP and the TTCN-3 as a means to depict test cases. The UML SD has been adapted and formalized to meet the requirements of PLC runtime systems in order to make them executable in previous works by Kormann et al. (2012) (*R1*). In this section the structure of the generated test cases in the SDs is discussed. Each test case contains three test components. For the test case shown in this paper (shown in Fig. 4) the three test components are: the *StampingPlantControl* which represents the system under test (SUT), the *Testsystem* which runs the test script and initiates the FI and the *StampingPlant* itself. Each test case consists of the following three parts (shown in Fig. 4, compartments marked on the right): 1) a "Start & Init" part *TC1_START_INIT()* which checks whether its Precondition is fulfilled and, as soon as the condition is met, initializes the test case; 2) a part, which starts the FI and verifies whether the observed software behavior is identical to the behavior specified and checks the termination criterion; 3) a completion part which logs, resets and prepares the plant for the next test case.

The Preconditions from the specification correspond to the states of the plant and are used for the generation of the "Start & Init" parts of the test cases. After the fulfillment of the Precondition, the FI is initiated and the test of the behavior of the system is started (observation of the global variables: alarm message *gStampCylError* and *gStampStop* specified in TSD Fault Class 1, which is not shown here). The termination criteria which is the duration in which a reaction of the end position sensor is expected and which may not exceed 0.5 seconds, and the maximum duration of the execution of the test case of 30 seconds are tested (Fig. 4). Once a test case has been completed, the results are logged, all modules of the sorting plant are stopped and re-initialization of the entire system is performed (completion part Fig. 4).

5. IMPLEMENTATION AND EVALUATION

The approach was evaluated for a scenario that uses the application example illustrated in section 4. The PLC

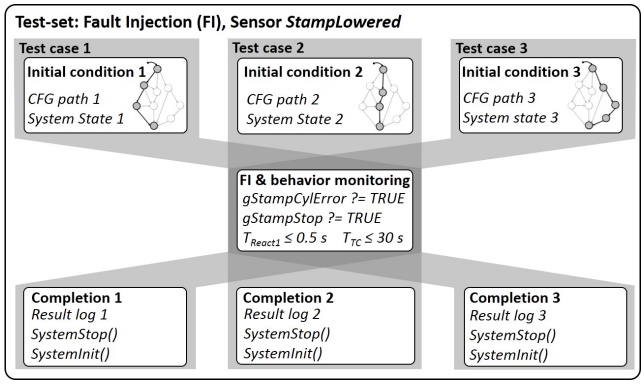

Fig. 5. Test set for FI *StampLowered*: test cases for different CFG paths

program of the plant was implemented in structured text (ST), as we have, up to now, implemented the extraction of the control flow for this language only. The information from the TSD and the FMEA was extracted manually solely from the information provided by the model specification of the developed model (*R2*) into the SD (*R4*) shown in Fig. 4 and then transferred manually accordingly into a test code on the CODESYS platform. In the example presented, a passive approach to execute the test cases was chosen. This means that the fault is injected as soon as the Precondition is met during the execution in the automatic mode of the plant (part 1 of the test cases). The Preconditions are checked in each cycle before the actual application is executed (Fig. 5, *R1*). After the Precondition for a test case is detected, the FI is initiated and the reaction of the plant is monitored (part 2 of the test cases). After completing the second part of the test case the result is logged and the system is stopped and then restarted (part 3 of the test cases). The executed test case is marked as tested and its Precondition will no longer be checked by the test driver. These steps are repeated until all test cases from the generated test set are marked as tested.

In the test run all Preconditions for the three generated test cases were found and all test cases were executed. During the execution of test case 1 the injected sensor failure of the end position sensor *StampLowered* was not detected within 0.5 seconds and the test case ended with the result FAIL. The CFG path was tracked back to the initialization run of the system. After analyzing the source code, which is responsible for the initialization run, the lack of error detection was found and the appropriate error handling routine could be added. During the execution of test cases 2 and 3 the injected sensor failure was detected and the test cases ended with the result PASS. The scenario showed that a software-implemented fault injection can be realized for PLC platforms and thus missing error handling routines of component failures (*R3*) can be identified in this way.

6. CONCLUSION AND FUTURE WORK

In this paper the requirements on a test case generation approach based on an industry research and analysis of industry code in the field of plant automation were stated. A model-based approach to test error handling routines implemented for component failures and the according

reaction of the PLC software was further on suggested using TSDs and FMEA analysis as a basis. It could be shown that such an approach is successful using a small laboratory plant, where sensor failures were injected and the reaction of the plant could be recorded. The FMEA introduces the means to make this approach scalable for plant manufacturers as possible component failures may be prioritized according to their respective risk priority number. Another advantage of the FMEA is that it can be used in order to document test coverage of faults and the robustness of a plant. The generated test cases are visualized by SDs. By using this specification the test engineers are supported in understanding and manipulating the executed test cases. By modeling different aspects of the plant behavior in TSDs and riks of faults in the FMEA and the combination of the generated SD test cases, a holistic model of possible faults with their related test cases which are already prioritized is achieved.

Still a lot of work remains to be done as already hinted at in section 2 and 5. The extraction of the CFG from the different languages of the IEC 61131-3 other than ST is one important step. As the work of deriving the test cases in this approach is done manually so far, the implementation of the adapted UML-PA diagram with the according test case generation algorithm is another important step. As the SD plugin is implemented in order to actively start testing sequences and for unit testing right now, it is also going to have to be adapted in conformation to our testing approach to constantly check for the Preconditions and start test sequences when they are met. To achieve this, it is planned to introduce an overlying test function in the code generated from the SDs which checks the execution for several Preconditions and as soon as one of the defined Preconditions is reached, the specific test case is started. In future work it will also be aimed at testing more complex scenarios such as multiple failures which lead to different alarms when combined. To analyze effects of several failures in combination a fault tree analysis is going to be analyzed. As mentioned in section 2 different faults such as faulty operator behavior, etc. also still have to be analyzed and strategies to test systems' reaction on their occurrence devised. In order to do this, models and notations have to be analyzed on their expressiveness in respect to the different fault categories. If a suitable model is found test cases may be derived from them in order to inject these kinds of faults.

REFERENCES

Baker, P., Dai, Z.R., Grabowski, J., Haugen, O., Lucio, S., Samuelsson, E., and Williams, C.E. (2004). The UML 2.0 testing profile. *Proc. of the Conf. on Qual. Eng. in Softw. Techn.*, 181–189.

Basile, F., Chiacchio, P., and Grosso, D.D. (2009). A two-stage modelling architecture for distributed control of real-time industrial systems: Application of UML and Petri Net. *Computer Standard & Interfaces*, 31(3), 528 – 538.

Biallas, S., Brauer, J., and Kowalewski, S. (2012). Arcade.PLC: A verification platform for programmable logic controllers. In *IEEE/ACM Int. Conf. on Automated Softw. Eng. (ASE)*, 338–341.

Feyerabend, K. and Josko, B. (1997). A visual formalism for real time requirement specifications. In *Transformation-Based Reactive Systems Development*, 156–168. Springer.

Hametner, R., Winkler, D., Östreicher, T., Biffl, S., and Zoitl, A. (2010). The adaptation of test-driven software processes to industrial automation engineering. In *IEEE Int. Conf. on Ind. Inf. (INDIN)*, 921–927.

Hussain, T. and Frey, G. (2006). UML-based development process for IEC 61499 with automatic test-case generation. In *IEEE Conf. on Emerg. Techn. and Fact. Autom. (ETFA)*, 1277–1284.

Katzke, U. and Vogel-Heuser, B. (2007). Combining UML with IEC 61131-3 languages to preserve the usability of graphical notations in the software development of complex automation systems. 10, 90–94.

Kormann, B., Tikhonov, D., and Vogel-Heuser, B. (2012). Automated PLC Software Testing Using Adapted UML Sequence Diagrams. *IFAC Symp. of Inf. Contr. Probl. in Manufacturing*, 14, 1615–1621.

Kormann, B. and Vogel-Heuser, B. (2011). Automated test case generation approach for PLC control software exception handling using fault injection. In *IEEE Ind. Electronics Society, IECON*, 365–372. IEEE.

Krause, J., Herrmann, A., and Diedrich, C. (2008). Test case generation from formal system specifications based on UML State Machines. *atp international*, 1, 47–54.

Kumar, B., Czybik, B., and Jasperneite, J. (2011). Model based TTCN-3 testing of industrial automation systems - First results. In *IEEE Conf. on Emerg. Techn. and Fact. Autom. (ETFA)*, 1–4.

Legat, C., Folmer, J., and Vogel-Heuser, B. (2013). Evolution in industrial plant automation: A case study. In *IEEE Ind. Electronics Society, IECON*.

Prähofer, H., Angerer, F., Ramler, R., Lacheiner, H., and Grillenberger, F. (2012). Opportunities and challenges of static code analysis of IEC 61131-3 programs. In *IEEE Conf. on Emerg. Techn. and Fact. Autom. (ETFA)*, 1–8.

Preusse, S. and Hanisch, H.M. (2008). Specification and verification of technical plant behavior with symbolic timing diagrams. In *3rd Int. Design and Test Workshop (IDT)*, 313–318.

Provost, J., Roussel, J.M., and Faure, J.M. (2011). Translating Grafcet specifications into Mealy machines for conformance test purposes. *Control Eng. Practice (CEP)*, 19(9), 947–957.

Reimann, G. (2013). Trendstudie: IT und Automation in den Produkten des Maschinenbau bis 2015. URL `http://www5.vdma.org/`.

Schlingloff, H. and Vulinovic, S. (2005). Model based dependability evaluation for automotive control functions. In *Modeling and simulation for public safety*.

Svenningsson, R., Vinter, J., Eriksson, H., and Törngren, M. (2010). MODIFI: a MODel-implemented fault injection tool. In *Computer Safety, Reliability, and Security*, 210–222. Springer.

Vyatkin, V. and Bouzon, G. (2008). Using visual specifications in verification of industrial automation controllers. *EURASIP Journ. on Embedded Systems*.