Model Based Specification, Verification, and Test Generation for a Safety Fieldbus Profile

Jan Krause[1], Elke Hintze[1], Stephan Magnus[1], and Christian Diedrich[2]

[1] Institut für Automation und Kommunikation (ifak),
Werner-Heisenberg-Str. 1,
39106 Magdeburg,
Germany
[2] Otto von Guericke Universität Magdeburg,
Lehrstuhl Integrierte Automation,
Universitätsplatz 2,
39104 Magdeburg,
Germany
{jan.krause,elke.hintze,stephan.magnus}@ifak.eu,
christian.diedrich@ovgu.de

Abstract. This paper suggests methods, and a tool chain for model based specification, verification, and test generation for a safety fieldbus profile. The basis of this tool chain is the use of an UML-profile as a specification notation, a simple high level Petri net model called "Safe Petri Net with Attributes" (SPENAT) and analysis methods found in Petri net theory. The developed UML-profile contains UML *class diagrams* and UML *state machines* for specification modeling. Verification and developed test generation methods are shown to be applicable after mapping the specification model onto SPENAT. The practical use of this tool chain is exemplarily demonstrated for a safety fieldbus profile.

Keywords: model based testing, verification, model based specification, SPENAT.

1 Introduction

More and more safety-relevant applications are being handled within industrial automation. The IEC 61508 standard describes requirements of functional safety. Microprocessor based device solutions for safety-relevant applications are faced with this standard. This forces the device manufacturer to contact third party partners such as TÜV and IFA which verify the development process and the development result. This results in a resource overhead for the device manufacturer. Therefore, these manufacturers are looking for methods and tools to automate some activities in order to decrease the overhead.

The paradigm of the model based system development (see e.g. [1]) is generally accepted handling the increasing complexity of the system and device development.

F. Ortmeier and P. Daniel (Eds.): SAFECOMP 2012, LNCS 7612, pp. 87–98, 2012.
© Springer-Verlag Berlin Heidelberg 2012

One usage of model based techniques is within the development of safety relevant fieldbus profiles in the industrial communication area. A fieldbus profile specifies the common use of communication services and interacting variables of selected device classes. These profiles serve as a basis for automation device development and are subject to certification tests in the framework of the related communication market organizations - the so-called user organizations. Devices which have successfully passed the tests can work interoperably if the coverage of test cases meets the necessary requirements. Additionally, the profile specification is part of a general quality process both within the user organization as well as the device manufacturer.

Using model based specifications as a result of profile development processes some quality assurance activities are addressed. One activity is the verification of syntactic and semantic correctness with regard to the specified requirements. Another activity is the generation of test cases with high specification coverage based on profile specification model.

To support formal verification and test generation from model based specifications a simple and intuitively understandable new Petri net model ("Safe Petri Net with Attributes" - SPENAT) was developed based on safe place transition nets (PT nets). Thanks to the simplicity of SPENAT a wide spectrum of existing and future modeling notations should be supported and usable for verification and test generation. The mapping of a UML State Machine to an ESPTN, the predecessor model of the SPENAT, is described in detail in [6].

In this paper methods for model based specification, verification, and test generation are introduced. All methods are implemented on a tool chain. The practical usage of this tool chain will be demonstrated on an existing safety fieldbus profile.

This paper is structured as follows. Section 2 addresses fieldbus profiles and their model based specification and section 3 introduces SPENAT, and discusses its verification and test generation. The methods introduced are implemented on a tool chain in section 4 and a case study is carried out for a PROFIsafe PA profile in section 5. Finally, section 6 concludes the paper and gives an outlook of future research.

2 Model Based Specification of Fieldbus Profiles

Device profiles usually provide variables and/or application functions with related input and output variables, parameters, and commands. In some cases, functions can be aggregated to function blocks. The variables, parameters and commands (called variables within this paper) are data to be communicated. The variables are dedicated to modules and sub-modules which provide addressing and data type information for the related communication services.

UML [14] nowadays is well established in the domain of embedded systems. Automation devices are seen as such systems. Class diagrams and state machine diagrams are the only UML languages which are used in the context of device and profile models. Class diagrams are used to describe the device structure which consists of functional elements and variables. A semantic enrichment of the classes is necessary which is done by the UML extension mechanism using stereotype and tags. The ste-

reotypes correlate with the device model and the class tags define the attributes of the profile elements - for instance the characteristics of the variables.

Table 1. Mapping of device and profile model elements to UML

Device and profile model element	UML language elements
Device	Class stereotype <<Device>>
Module	Class stereotype <<Module>>
Function Block	Class stereotype <<FB_Type>>
Physical Block	Class stereotype <<PB_Type>>
Transducer Block	Class stereotype <<TB_Type>>
Function	Class stereotype <<Function_Type>>
Variable	Class stereotype <<Variable_Type>>
Attributes of Variable	Tagged Value of class stereotype <<Variable>>
Behavior of function blocks	State machines

The result is a UML profile template with the standard elements which can be used by the profile developers. Table 1 gives an overview of all used model elements and their UML representation.

In order to generate test cases and/or to verify the fieldbus profile model in view of safety relevant properties such as deadlock freeness and/or reachability analysis of special states Petri net methods can be used. Therefore, a mapping of this fieldbus profile model onto SPENAT needs to be implemented (see [6]). The Petri net dialect SPENAT as an extension of the Petri net dialect from [6] is introduced in the next section.

3 Safe Petri Net with Attributes (SPENAT)

3.1 Motivation

The SPENAT notation is built upon safe place transition nets (p/t net) [10] and concepts of high level Petri nets [3], [4], [5], [10]. Using SPENAT it is possible to use external and parameterized signal/events as transition triggers (in contrast to STG [11], SIPN [15], IOPT [16]). Thanks to this feature it is much easier to model the required behavior of an open and reactive system with a Petri net. Also, the mapping of existing models onto a Petri net should be possible in an easy and intuitive way.

An example of a declaration of a Petri net reacting on externally parameterized signals is presented in Fig. 1. This Petri net has two transitions where transition $\mathbf{t_2}$ can only fire after transition $\mathbf{t_1}$ and the guard of $\mathbf{t_2}$ depends implicitly on the value of the parameter **x** of the trigger event of $\mathbf{t_1}$.

If transition $\mathbf{t_2}$ of the Petri net of Fig. 2 fires, it is clear that the parameter **x** of the external event **ev1(int x)** must be **1**. This value is a result of the guard of $\mathbf{t_1}$ (**msg.x<2**), the effect of $\mathbf{t_1}$ (**y=msg.x**), and the guard of $\mathbf{t_2}$ (**y>0**). The keyword **msg** is a reference to the respective trigger event of the transition. In this case the value **1** is

the only valid value for parameter **x** of the trigger event **ev1(int x)** so $\mathbf{t_2}$ can fire. For any other value of **x,** transition $\mathbf{t_1}$ cannot fire (see guard **msg.x<2**) otherwise the SPENAT of Fig. 1 would be in a deadlock after $\mathbf{t_1}$ has been fired.

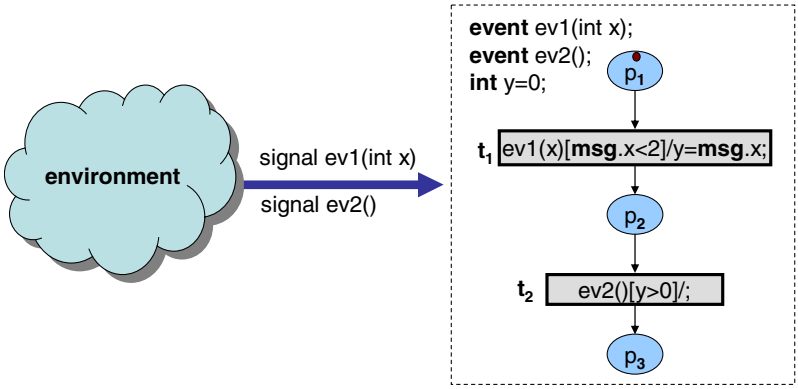


Fig. 1. SPENAT with externally parameterized signals/events

3.2 Structure and Behavior

Like every Petri net SPENAT is characterized by a bipartite graph. The places of SPENAT are typecasted (see [3], [5], [10]). The transitions have some special properties defining their firing behavior. The markings of a SPENAT are distinguishable. Places of SPENAT can only be marked with one (colored) token at the same time and the arcs are not inscripted. Furthermore, transitions can fire if an external parameterized event appears and there is a clear separation between control and data places. Control places can only be marked with the token • and data places can only be marked with a colored token [3]. Hierarchies are not allowed within SPENAT.

The color of a colored token of a data place represents a data value. Every data place belongs to the initial marking $\mathbf{M_0}$ of SPENAT. Data places represent the attributes of SPENAT and it is mandatory that every attribute has an initial value.

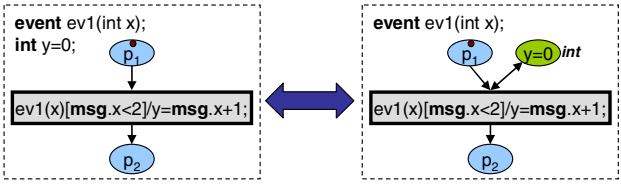


Fig. 2. Declaration of SPENAT

In Fig. 2 an example of SPENAT is outlined. On the left the attribute **y** of SPENAT is declared, whereas on the right the equivalent net representation without

an explicit attribute declaration is outlined. Furthermore, SPENAT can receive the external event **ev1** with one parameter **x** of type **int**.

Based on Fig. 2 the essential properties of SPENAT can easily be identified. The connection of a data place to a transition is always implemented by a loop, so every data place which is a predecessor of a transition is always a successor of the same transition. Whether a data place is connected (by a loop) to a transition is determined by the transition inscription (guard and effect). With this property and the fact that data places are part of the initial marking, data places are always marked. This restriction allows a more simplified analysis of SPENAT. Also, the declaration of the data places is not mandatory for the graphical declaration of SPENAT (see Fig. 2).

The syntax and semantic of the inscription of a SPENAT transition is essentially adequate (see Fig. 2) to the syntax and semantic of a transition of a UML State Machine (USM [14]). However, a transition of SPENAT can have more than one predecessor which is not possible for a USM transition. A SPENAT transition fires if all predecessors are marked, if its (external) event (its trigger event) appears, and if its guard is evaluated as 'true'. If a transition fires, all specified actions associated with the effect of the transition are executed.

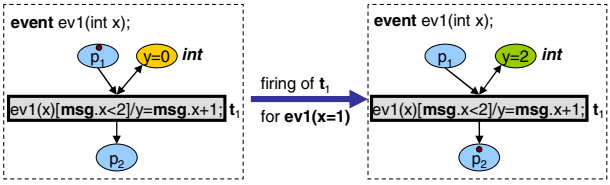


Fig. 3. SPENAT before and after the firing of a transition

In Fig. 3 SPENAT with a data place **y** is presented before and after an firing of a transition ($\mathbf{t_1}$). The places $\mathbf{p_1}$ and $\mathbf{p_2}$ are control places. The initial marking $\mathbf{M_0}$ of this SPENAT is characterized by the set $\mathbf{M_0=\{(p_1,\bullet),(y,0)\}}$. The marking $\mathbf{M_1=\{(p_2,\bullet),(y,2)\}}$ is induced by the firing of $\mathbf{t_1}$ based on $\mathbf{M_0}$. A marking set M contains all current colored tokens. Here, a colored token represents a pair of place and value (color, see [3]). The element • is used as a type and a value of control places. More than one colored token cannot be used in the current marking set for one place, so SPENAT is safe.

If a transition of SPENAT fires, all colored tokens representing a predecessor of the transition will be removed from the current marking set and for each successor a new colored token is produced and added to the current marking set.

3.3 Verification of SPENAT

For an exhaustive analysis of SPENAT a state space analysis is necessary. The state space of a Petri net can be represented by its reachability graph. For the creation of the reachability graph of a Petri net all possible processes are sequentialized, which is a main drawback of this state space coding. The state space explodes if the Petri net is

strongly concurrent. Therefore, it is suggested to use the complete prefix of the unfolding [7], [9], [11] of the Petri net for analysis tasks.

There are several algorithms for creating the prefix of the unfolding of a Petri net, most of them for a safe place transition net. The first algorithm was developed in [9]. This algorithm was improved in [7] by the use of a total order for the prefix events for the construction of a minimal prefix. In [11] this algorithm was parallelized and further optimized. Also, in [11] the dependence on general place transition nets was removed. Thus, the algorithm for prefix construction now is applicable to higher Petri nets as well.

Values of attributes of SPENAT can depend on values of external event parameters. In order to represent a marking of SPENAT during the prefix calculation algorithm classic (value based) marking representation of (colored) Petri nets are not suitable. However, the marking of SPENAT attributes can be expressed by a set of constraints. A marking of SPENAT can then be represented by a marking set for the control places and by the identified constraints for the data places. With this marking representation and the results of [11] the known algorithms for the prefix creation can also be used for the prefix construction for SPENAT. However, the method for the extension of the prefix with new events has to be adapted because of the use of constraints as marking representation for a data place. Now a new event can only be added to the prefix if the identified constraints are satisfiable. Also, the identification of the cutoff events has to be adapted. Now it is necessary to check if two events produce the same marking of control places as well as the same constraints on a semantic level.

In Fig. 4 SPENAT with its prefix is presented. The events of the prefix are inscripted with the constraints for the data places. The parameters of the trigger events are associated with the respective prefix events for a better overview. The constraint $\mathbf{e_1.x<10}$ of the event $\mathbf{e_3}$ is valid for the parameter **x** of the trigger event **ev1(int x)** of transition $\mathbf{t_1}$ represented by $\mathbf{e_1}$ in this process. The prefix contains the cutoff events $\mathbf{e_2}$, $\mathbf{e_7}$, and $\mathbf{e_8}$. All cutoff events correspond to the marking of event $\mathbf{e_1}$. Furthermore, a deadlock can be identified within the prefix seen in Fig. 4. This deadlock is a result of the execution sequence $\mathbf{t_1t_3t_5}$ represented by the local configuration $\mathbf{\{e_0,e_1,e_3,e_5\}}$ assigned to the prefix event $\mathbf{e_5}$ (not added here).

In general, the complete prefix of the unfolding of Petri net is a compact representation of the state space and is well suited for the verification of interesting properties like deadlock and reachability analysis, and satisfiability of LTL formulas by methods of model checking. In [8] and [11] methods for formal verification based on prefix are presented. These methods are also applicable to the verification of SPENAT.

3.4 Test Generation Based on a SPENAT Specification

The method of the test generation is also based on the computation of the prefix of SPENAT. This method is described in [6] in more detail. The work in [6] was extended in such a way that external events with parameters can now be used, too.

The steps of the test generation based on SPENAT specifications are straight forward. First the prefix is constructed based on the specified test criteria (e.g. coverage criteria). It is not strictly necessary to construct the complete prefix of the unfolding of SPENAT. If all places are covered by at least one test case the resulting prefix is in general much smaller than the complete prefix.

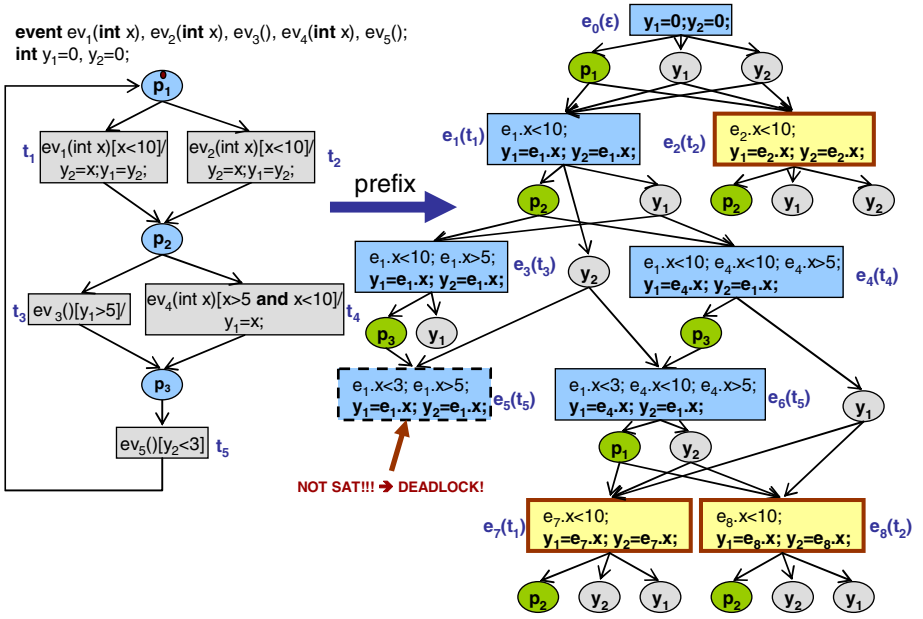


Fig. 4. SPENAT and its prefix

After the prefix construction the test cases are identified. The prefix of the unfolded SPENAT is an acyclic Petri net in which all possible processes of SPENAT are contained within the prefix. Each possible process can be identified by a prefix event or rather by the local configuration of a prefix event [7]. Prefix events with no successor events and a maximal number of transitions represent *maximum processes*. So in general, it is a good strategy to associate each identified maximum process with a test case. With this strategy the coverage criteria "round trip path" of SPENAT can be achieved.

The values of the external events as the stimulus of the test object are constrained by the inscription of the prefix events. When instantiating a process and assigning it to a test case, a value within the specified value range needs to be selected. This can be carried out in a random way but in general it is a widely accepted strategy to select a bound (upper and/or lower) within the specified value range.

The identified test cases specify a (concurrent) message exchange between the test object (System Under Test – SUT) and the tester or test system. This is an abstract sequence-based description of the stimuli and the expected responses of the test object. This abstract representation of the test cases must be transformed in an understandable and executable format for the test system. Furthermore, the realized level of abstraction during the modeling of the required test object behavior must be respected in order to get automatically executable test specifications as a result of the test generation process.

Data types, events, and/or signals, modelled within the profile model at an abstract level, have to be mapped to usable structures of the target test notation of the used test

tool. Therefore, rules are necessary in order to automate this test formatting. For the formatting in the standardised test notation TTCN-3 [12] and in a proprietary test notation based on C#, suitable rules were developed and are implemented.

4 Tool Chain

The previously described methods are implemented with established tools and prototypical implementations (see Fig. 5) on a tool chain. For the modeling, the established UML modeling tool Rhapsody by IBM is used. The modeling of the profile specification is done using Rhapsody and our definition of the UML profile.

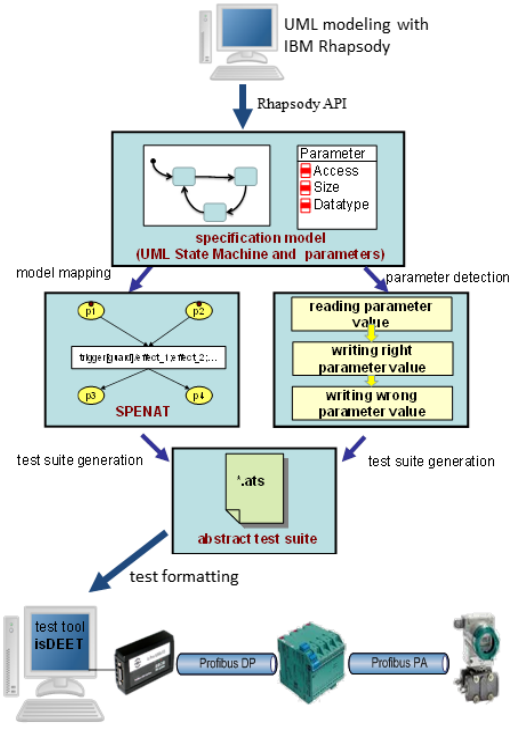


Fig. 5. Tool chain for test generation and test formatting

With the available API of Rhapsody, the model of the profile specification can be extracted. Therefore, two different test generators can operate using this profile model. The test generator for dynamic specification elements searches for UML state machines and generates, for these models of the expected behavior, suitable test cases with Petri net techniques (see left branch of Fig. 5) as described in this paper. The test generator for the parameter testing looks for special stereotypes within the profile model indicating the parameter classes (see right branch of Fig. 5). This test generator is not discussed in this paper.

The result of the two test generators are abstract test cases on the same level as the specification model. These abstract test cases have to be transformed into suitable test notations in order to automatically execute the tests with suitable test tools. This transformation is implemented for TTCN 3 and a C# based test notation for the used test tool isDEET. The test execution and test verdict identification can be realized with isDEET.

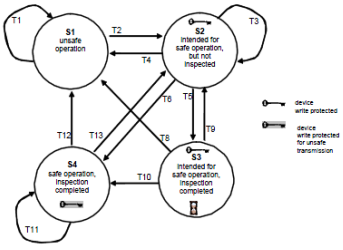
5 Case Study for the PROFIsafe PA profile

The UML-profile previously described was used within a project to describe the structure and the required behavior based on the specification of the PROFIsafe PA profile (see extracts in Fig. 6). The required behavior was modeled with a UML State Machine. Then, model verification was done in a first step to guarantee that the model was free of errors. In particular, safety critical properties like deadlock freeness and reachability of all states and all transitions were checked.

Based on the verified model of the specification of the PROFIsafe PA profile, a test suite with high coverage (coverage criteria "round trip path") was generated. The generated test cases were transformed from the abstract sequence based format to the input format of the used test tool (based on C#). Variables necessary to influence the state machine and get the state machine status are additionally used for this transformation. For this transformation some rules had to be implemented by an adapter in order to handle the actual communication between the test tool and the test object. A SIEMENS device ("SITRANS P") was successfully used as a test object. Except for the implementation of the transformation rules by an adapter for the test cases of the test tool, all activities were executed automatically.

Name	Data Type	Access	Size
DEVICE_SIL	Unsigned8	r	1
WRITE_LOCKING	Unsigned16	r,w	2
DESCRIPTOR	OctetString	r,w	32

(a) parameter characteristics



(b) PROFIsafe state machine

Fig. 6. (a) Three selected parameters and (b) The state machine of the PROFIsafe profile [13]

5.1 Specification Model of the PROFIsafe PA Profile

The profile specification for PROFIsafe PA [13] includes the description of the behavior and PA-PROFIsafe specific profile parameters. For example, the state machine and a selection of three parameters are shown in Fig. 6. Before starting the communication, a device must go into the safe state "S4". In the state "S1" a standard unsafe

communication is still possible. The changeover to error-free communication is reached via the states “S2” and “S4” or “S2”, “S3” and “S4” (see Fig. 6).

All PROFIsafe profiles consist of various blocks such as “physical block“, “function block,” and “transducer block.” The different blocks are specified in detail with functional elements. Thus, in each block the corresponding profile parameters are defined as class attributes. Additionally, a parameter class for each parameter was created with the parameter characteristics as tags.

5.2 Verification and Test Generation

A tool developed at the ifak Magdeburg is used to verify the specification model and generate test cases out of this verified specification model. It allows transferring a UML state machine into SPENAT, and it creates the complete prefix of the unfolding of this SPENAT. Based on this prefix the verification (deadlock and reachability analysis) and the test generation are done. Therefore, different structural coverage criteria can be chosen. For the highest possible coverage criteria “round trip path” 52 test cases are generated.

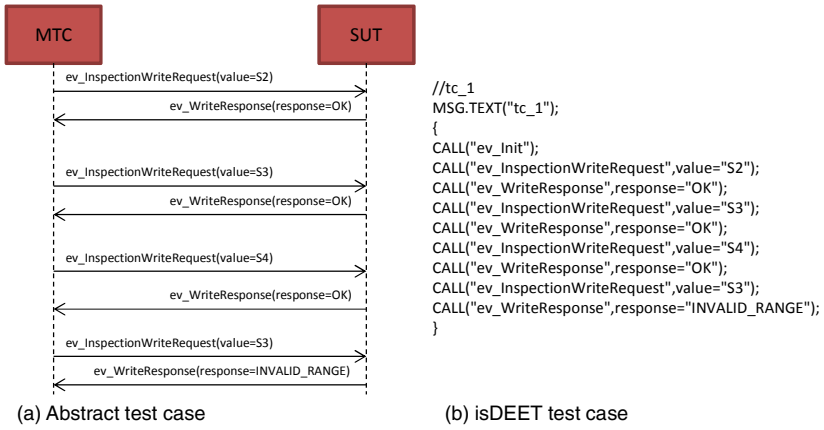


Fig. 7. Abstract and formatted test case for state machine testing

Fig. 7 shows an example of a test case as a sequence diagram. The test case will run through all four states of the PROFIsafe state machine starting in state “S1”. State “S2” is initiated by a write request on the inspection parameter and confirmed with a positive response. The transition to “S3 and then to “S4” takes place in the same manner. Finally, an attempt to execute a transition from state “S4” to state “S3” is made. According to the state machine, this is an illegal transition and a negative response is returned by SUT.

5.3 Test of the PROFIsafe Device Siemens SITRANS-P

Fig. 8 shows the test setup with the test device Siemens SITRANS-P. The test tool "isDEET" runs on a computer which can access the PROFIBUS devices via the "is Pro Profibus USB Interface". Using a segment coupler the SITRANS-P device is connected to PROFIBUS PA.

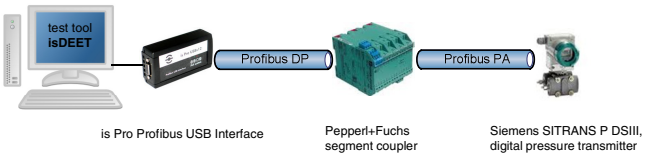


Fig. 8. Test setup

All generated abstract test cases are transformed into an executable test notation and afterwards run as a combined test suite on the test system. The test tool creates a report of the success or failure of the executed test cases. The testing of parameter and state machine test cases for the PROFIsafe profile was successful. The result of the test suite confirms the correctness of the device regarding the profile on the one hand, where functionality and behavior comply with the profile and its requirements. On the other hand, a successful validation of the method for test case generation and the transformation in the test notation are shown with the established test device used.

6 Conclusion and Outlook

In this paper an approach to model based specification, verification, and test generation for safety fieldbus profiles were introduced. The essential methods and tools ranging from model based fieldbus profile specification to the test execution are described. Here, UML was used for the fieldbus profile specification, and Petri net methods were employed for the model verification and test generation. The developed Petri net model "Safe Petri Net with Attributes" (SPENAT) was used for the mapping of the UML model and the application of the Petri net methods. The practical use of these methods was demonstrated with an existing safety relevant UML profile (PROFIsafe PA profile) for fieldbus devices in the PROFIBUS and PROFINET domain.

In the future, more existing methods of formal verification from the petri net area should be used to verify the SPENAT model. Especially model checking algorithms should be applied for the SPENAT analysis. Additionally, the method for test case generation should be more configurable. One goal is to have more possibilities for controlling the test generation process. The description of distributed (cooperative) systems with communicating SPENAT components is an ongoing future research aspect. The verification and generation of tests based on domain specific models will gain an increasing importance in the future for distributed and cooperative systems.

References

1. Schätz, B., Pretschner, A., Huber, F., Philipps, J.: Model-Based Development of Embedded Systems. In: Bruel, J.-M., Bellahsène, Z. (eds.) OOIS 2002. LNCS, vol. 2426, pp. 298–311. Springer, Heidelberg (2002)
2. Frenzel, R., Wollschlaeger, M., Hadlich, T., Diedrich, C.: Tool support for the development of IEC 62390 compliant fieldbus profiles. In: Emerging Technologies and Factory Automation (ETFA), IEEE Conference (2010)
3. Jensen, K.: Coloured Petri Nets: Modeling and Validation of Concurrent Systems. Springer, Berlin (2009)
4. Best, E., Fleischhack, H., Fraczak, W., Hopkins, R., Klaudel, H., Pelz, E.: A Class of Composable High Level Petri Nets. In: ATPN 1995. Springer (1995)
5. ISO/IEC 15909-1: Software and system engineering – High-level Petri nets – Part 1: Concepts, definitions and (2004)
6. Krause, J., Herrmann, A., Diedrich, C.: Test case generation from formal system specifications based on UML State Machines. atp - International 01/2008. Oldenbourg-Verlag (2008)
7. Esparza, J., Römer, S., Vogler, W.: An Improvement of McMillan's unfolding algorithm. In: Formal Methods in Systems Design, vol. 20, Springer (2002)
8. Heljanko, K.: Combining Symbolic and Partial Order Methods for Model Checking 1-safe Petri Nets, PhD thesis. Helsinki University of Technology, Helsinki (2002)
9. McMillan, K.L.: Using Unfoldings to avoid the State Explosion Problem in the Verification of Asynchronous Circuits. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 164–177. Springer, Heidelberg (1993)
10. Girault, C., Valk, R.: Petri Nets for Systems Engineering: A Guide to Modelling, Verification, and Applications. Springer, Heidelberg (2003)
11. Khomenko, V.: Model Checking Based on Prefixes of Petri Net Unfoldings. University of Newcastle (2003)
12. ETSI: Testing and Test Control Notation (2009), `http://www.ttcn3.org/`
13. PNO, PROFIBUS Specification: PROFIsafe for PA Devices. V1.01 (2009)
14. Object Management Group: Unified Modeling Language 2.2 Superstructure Specification (2009), `http://www.uml.org/` (January 08, 2010)
15. Frey, G.: Design and formal Analysis of Petri Net based Logic Controllers, Dissertation. Shaker Verlag, Aachen (2002)
16. Gomes, L., Barros, J., Costa, A., Nunes, R.: The Input-Output Place-Transition Petri Net Class and Associated Tools. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN 2007), Vienna, Austria (2007)